

SINCLAIR



WORLD

EVERY MONTH

FEBRUARY 1991

£1.75

2	9	14	26	46
4	10	15	34	48
8	12	16	36	49
		18	43	
		22	44	

QL

QLIPBOARD

DIY Toolkit's recipe
for moving text



Product review:

PC KEYBOARD 90

hardware interface

PUTTING THE BOOT IN

Super Basic tackles
the start-up syndrome

30 THINGS TO DO WITH YOUR QL



SINCLAIR



Editor
Helen Armstrong

Production Controller
Jayne Penfold

Designer
Jeff Gurney

Advertising Sales
Jason Newman

Magazine Services
Sheila Baker

Advertising Production
Michelle Evans

Group Advertising Manager
Jean Dorza

Group Editor
John Taylor

Publishing Director
Ray Lewis

Managing Director
Peter Welham

Sinclair QL World
Panini House
116-120 Goswell Road
London EC1V 7QD
Telephone 071-490 7161
ISSN 026806X

Unfortunately, we are no longer able to answer enquiries made by telephone. If you have any comments or difficulties, please write the The Editor, Open Channel, Trouble Shooter, or Psion Solutions. We will do our best to deal with your problem in the magazine, though we cannot guarantee individual replies.

Back issues are available from the publisher price £2 U.K., £2.75 Europe. Overseas rates on request.

Published by Maxwell Specialist Magazines, A Division of MCPC Ltd., Sinclair QL World is distributed by SM Distribution.

Tel: 081-677 8111. Subscription information from: MSM Subscription Dept, Lazahold Ltd., PO Box 10, Roper St, Pallion Ind. Est., Sunderland SR4 4SN.

£21.00 U.K., £24.70 Europe, MiddleEast £25.80, Far East £27.60, Rest of World £26.20. U.S.A. \$45.00.

Airmail rates available on request.

Typesetting by Ford Graphics, 8-10 Whitsbury Road, Fordingbridge, Hampshire. SP6 1BR. Tel: (0425) 655657. Printing by Cradley Print. (0384) 411411. Sinclair QL World is published on the third Thursday preceding cover date.

© COPYRIGHT
SINCLAIR QL WORLD - 1991

C O N T E N T S

■ ■ February 1991

- 9 QL SCENE ● New from Van der Auweras**
- 10 OPEN CHANNEL ● Designs for a new QL?**
- 12 TROUBLESHOOTER ● A first look at Perfection**
- 17 THE ITALIAN JOB ● The Lake Garda QL show**
- 18 SUPERBASIC ● Putting the boot in**
- 21 SOFTWARE FILE ● Double Block**
- 20 TECHNICAL HELPLINE ● Queries answered**
- 24 30 THINGS TO DO WITH A QL ● Just a few suggestions**
- 28 DIY TOOLKIT ● Qlipboard - part two**
- 33 SOFTWARE FILE ● Anelpum Quat**
- 34 PRODUCT REVIEW ● Falkenburg PC keyboard interface**
- 36 PROGRAMMING IN C ● The last part of our series**
- 44 SOFTWARE FILE ● File Executive**
- 48 QL TO PC FILE TRANSFER ● Hardware links**
- BACK COVER SUBSCRIPTION INFORMATION**



NEXT MONTH

THE USER GUIDE GUIDE

Mike Lloyd is working on the QL User Guide Guide and we now hope to have the first instalment for March. We hope that it will be a centre-spread pull-out as well, so that you can store it separately.

HARD DISK COMMENTS

New hard disk owner Bryan Davies shares his experiences so far with the modern medium.

QL

SCENE

New from Van der Auweras

Progs Van der Auwera have released a new program, *Qractal*, which (oddly enough) generates Mandelbrot or Julia set pictures on the QL. *Qractal* is described as featuring fast, precise 2^{28} fixed point calculations, toggling between a Mandelbrot set picture and its Julia equivalent; saving, redrawing, recoloring of maps 'to discover the hidden structure of the set', 100% machine code and multitasking program using pointer environment and menu extensions; save option for created screens; inner set filling with attraction count of value; option to change co-ordinates of Mandelbrot centre; supports BDM or binary decomposing, mode 4 and mode 8, variable maximum

number of iterations; easy zoom; special clear colouring method; option to make parameter files. The program comes on three disks with sample charts, and costs £40 sterling (including conversion costs) or Belgian francs 2000 (post free in EC, post BEF 300 outside EC area).

The Van der Auweras have also updated *The Painter* to V4.00. The cost of the update is £14 (BEF 800) without binder, £17 (BEF 1000) with binder. Add £10 postage/bank conversion charge if sending a Sterling cheque (BEF cheques do not incur the extra charge).

Progs Van der Auwera are at Haachtstraat 92, 3020 Veltem, Belgium. Tel (local) (016) 488952.

Changes in Quanta Editor resigns

The November issue of the user group Quanta magazine has some updates on sub-group information, and details and maps relating to the November workshop in Nottingham. Apart from that, the A5 magazine is mainly full of tips and responses from members. It's entertaining to see the familiar *DIY Toolkit* headline suddenly pop up at the back in a full-page list of Simon Goodwin's Toolkit disk collection, available from CGH Services.

The November editorial comments that The Consumer Protection Office in Birming-

ham now has a number of complaints against supplier PDQL, along with an odd comment about the office's "new address". The address — the same one that appeared in QL World October 1990 — is given as **155-157 Corporation Street, Birmingham B4 6TH.**

We note that the Essex sub-group has been 're-chaired' by Ron Dunnett. Ron is also trying to set up an 'out of Quanta' public domain library and would appreciate support.

As we write, we hear that Sarah Johnson has resigned as Quanta editor.

German Group

The Sinclair QL User Group Ev (SQLUC) of Germany has nearly 1,000 members, mainly in Germany but also in Japan and other parts of Europe and has been in action for nearly six years.

SQLUC publishes a quarterly magazine in German, *Quasar*, and holds local meetings monthly. "The QL is not at all dead, but remarkably

alive... anyone who roughly understands German should think about getting a membership in our non-profit organisation", says Foreign Contact Franz Herrman.

The Sinclair QL User Club EV are contactable via Franz at Talstrasse 21, D-W5460 Ochenfels, W. Germany. Tel. (local) (0) 2644-1855.

Draw

Congratulations to the following winners in the MSM November subscriptions renewal draw:

J A Pedder of Luton wins £150

A J Smith of Northampton wins £50

R D Knight of Essex wins £50

When subscribers renew, their names will automatically be entered in the monthly renewal draw. You can renew at any time. Call 091 510 2290. Good luck!

It's a Q

Olympic, publishers of *Goblin's Quest*, have written to say that the instructions for the program can now be read on televisions as well as monitors. Olympic also have a new telephone number (German local) 02134 96694.

We also apologise for printing Olympic's name as 'Olympic' — we had a flag out to catch that very obvious mistake, but it got through. We hope it won't happen again.

Olympic's address is **Olympic Computer Systems, Quellenweg 18, 4220 Dinslaken, W. Germany.**

New Sub-group

The first meeting of the new Quanta sub-group in London was held on August 1 at the Marquis of Clanrickad pub near Paddington Station. Tony Firshman of TF Services gave a demonstration of the QBox bulletin board, Mark Knight demonstrated his Molecular Graphics chemistry program (available from the Microdrive Exchange) and Laurence Reeves of QView gave a talk on Minerva.

The subject of the September meeting was QPAC2, the rewrite of Qram.

There will be a hands-on opportunity to try QPAC2. Each meeting will have a "beginners' corner".

Subsequent meetings will take place on the first Wednesday of each month. For information contact **Jeremy Davis, 6 Elmcroft Crescent, Harrow, Middx HA2 6HN. Tel. 081 863 1631.**

OPEN CHANNEL

Open Channel is where you have the opportunity to voice your opinions in *Sinclair QL World*. Whether you want to ask for help with a technical problem, provide

somebody with the answer, or just sound off about something which bothers you, write to: Open Channel, Sinclair QL World, 116/120 Goswell Road, London EC1V 7QD.

Mail

I'm a first-year computer science student at Brunel University, and I have seen appeals for QL news through Electronic Mail.

There must be a few more QL owners who are university based with access to a main-frame. If so I would welcome electronic mail from them at C589559 @ UK, ac, brunel, cct.

I'm in a sandwich course, so am going for industrial placement very soon but these computer-ese people are welcome to write to me at home.

Sunil Gupta
45 Marlborough Crescent
Grays
Essex

Survival

I have been going to write to you for some time to say that I think the QL World is a most excellent magazine.

It is perhaps invidious to pick out particular contributors but Mike Lloyd's series on Better Basic and SuperBasic have been immensely educative and enjoyable. Also, Simon Goodwin's DIY Toolkit has provided many helpful titbits along the way.

So glad to learn that the magazine has survived an upheaval that could have seen its demise.

J W Hill
Matlock
Derbyshire

Mail

I was interested to read Bryan Davies' comments about the upgrade paths (un)available to QL users (October 1990). As a QL owner since 1985, I was very interested to read about some of the ideas being thrown around at the moment.

Having already decided to buy another computer, I have been shopping around looking at IBM compatibles with the view to buying a '386 (SX) system for myself. However, I have yet to come across a system that gives me the flexibility of true concurrent programs (Yes, some of us do use it!) together with a very good interpretation of Basic that I currently have access to on my QL. I find myself in the position that I am not prepared to pay the sums of money required to buy a machine that, in almost every respect except speed, is inferior to my QL.

So, in answer to your solicitation for ideas for new machines, I have included my 'dream machine' spec.

In my sunny, happy (QL) world I would be able to buy a 'box', with say a ten slot backplane (obviously built to a non-standard spec!) and the following cards installed.

CPU Card: 68040 processor running at 25 MHz with an amount (expandable?) of cache memory. 2MB of SIMM memory onboard. QDOS+ multitasking operating system in ROM (basically a 32 bit amalgamation of QDOS, TK2, TK3 and DIY toolkit routines) including a 'TaskMaster' sort of program/multitasking/file manager shell. Keyboard driver chips.

Memory Card: An address selectable memory card expandable to 4MB per board

using standard SIMM's. Each system would be able to address a maximum of 7 such cards to give the possibility of 28MB of 'flat' expansion memory.

Disc Interface Card: Interface allowing connection of 2 Floppy drivers and a hard drive. Formatting, copying and backup routines contained on 'local' rom to allow true background task disc operations.

Video Board: Resolution around double that of the present QL display, and with an expansion of the palette.

Application Board: An EPROM/ROM allowing the installation of up to 16 application EPROMS which could then provide for embedded application multitasking. Workstations could be produced just using a floppy for data storage, all applications being on board and running at boot-up time.

Other boards could include, modems, real-world interface and development boards etc, etc. The use of a standard (sic) bus and the availability of blank boards could start many people experimenting furiously, not least of all myself. I have deliberately refrained from including a price range for this 'dream machine' because we all know that dreams do not come cheap. This, however, would be one dream that I would sell (almost) everything for!

As for all the redundant QL's, I'm getting interested in simple multiprocessor applications so I'll take them off everybody for a fair price, although I'm sure many other can think of things to occupy a QL put out to graze! Any takers for my dream?

Gareth Williams
Shepperton
Middx

Editor's notebook

As the new Year approaches, I contemplate the things that gone astray in the Christmas Rush – a page here, an author there, a supplier somewhere else. Why has Bill Richardson not collected his valuable monitor? What has become of Abacus article? Where is my letters file? How is Mike Lloyd to finish the first instalment of the QL User Guide for the March issue? Who plugged in the first fairy lights where my computer used to be?

Then I contemplate all the things that haven't gone astray.

Bryan Davies has had first try of Perfection and reports in Troubleshooter. Simon Goodwin has finished Qlipboard a valuable new function for the QL. Howard Clase has sent an edition of Technical Helpline all the way from Nova Scotia. Jeugen Falkenburg has produced a new PC keyboard interface. At last we have room to run the concluding part of Andy Wright's Programming in C. And much, much more.

1991 will be a time of more concentrated action on QL hardware than we have seen for a number of years. Interesting times indeed!

PRODUCT REVIEW

REALTIME CLOCK

The QL has always lacked a means of retaining the correct time and date when it is switched off. In the past, there have been one or two systems which have managed to overcome this short-fall; but with the passage of time, these have become more and more difficult to obtain. This is a relatively recent system, which comprises a small circuit board, a small rechargeable battery and a few more Basic commands to load from disk.

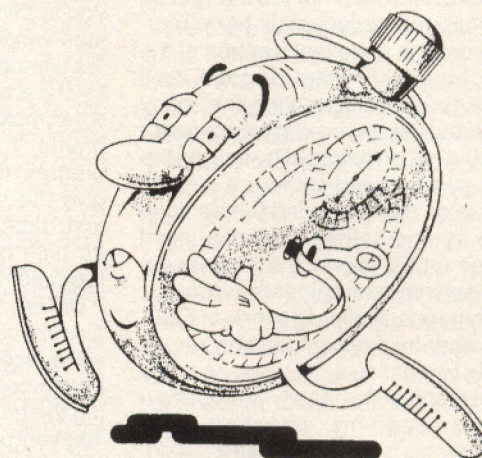
The hardware element is quite simple to install. You open your QL (by removing eight screws), and remove two chips: the 8301 (which is a large chip next to the 68008) and one of the two rom chips. Next you push the main circuit board firmly into the empty rom socket, and push a through connector into the 8301 socket. Now all you have to do is replace the two chips by placing the rom and the 8301 chip into the supplied circuit board and through connector respectively, decide where to place the battery (there is room next to the expansion socket), and screw the QL back together.

File

Users who already have something else in one of the rom sockets, such as *Minerva*, or the Qimi interface, should not experience any problems, since the real time clock will quite happily sit in either rom socket. I did however find that the board supplied and the *Minerva* circuit board formed a very tight fit. Although I was able to install the board without too much trouble, some users may find that they need to file a little bit off one of the boards.

The instructions for fitting the Clock board are very straightforward and should not cause any problems. One thing however that the manual omitted to mention was that if

The QL has always been short of a perpetual time and date function. Rich Mellor tries a battery-backed board which uses one of the rom sockets.



you are unsure which way around the chips faced, there is a small notch on them which should always be towards the rear of the QL.

Once installed, the QL clock can be set to the same date and time as that kept by the real-time clock by using one of the four supplied Basic keywords. Should the stored date and time ever be incorrect (for example if the battery is running low) then you can set the QL clock manually and then use another new keyword to write this value to the battery-backed clock. Using this method means that the time on boot-up cannot be corrupted accidentally by software which uses *SDATE* for its own purposes, nor should it be corrupted by the QL being switched on and off.

Recharge

One advantage of this package is that there is never any need to replace the battery, since it constantly recharges itself whenever the QL is switched on. It should be able to keep the time and date ac-

curately for up to one month without the QL ever being switched on, which should be more than ample for most QL users.

Extra

A small oddity with the package is that there is a spare 32 bytes of ram on board which is also preserved by the battery. This is accessed by using the other two new Basic keywords, but is so little as to prove not very useful, however the manual suggests that it could be used to store default baud rates or a default directory for *Toolkit II*. A more useful idea would be to use it to store a password for a software package; unless the unwelcome user knew that this memory existed, he would find it very hard to trace the password.

On the minus side, certain things plugged into the QL rom port can affect the real-time clock, since it is located in the last four addresses used by the rom port. This should not give many problems (certainly *Toolkit II*, *Ice* and *Lightning* all work okay), and

can easily be checked before ordering the package by the following lines:

```
10 FOR a=1 TO 4
20 PRINT PEEK (1024*64-a)
30 END FOR a
```

If the values returned are all either 0 or 255, then you will have no problems. In the manual Phil states that he may be able to adapt the package for use with problem roms, but this will only be effective on JM and *Minerva* QLs.

Overall the package is extremely useful and considering the cost of making a hardware add-on in small quantities, is quite cheap. When balanced by the time and effort required to set the QL clock each time the QL is reset, the package is a bargain.

INFORMATION

Product: Real Time Clock (V1.20)

Price: £29.50

Supplier: Phil Gaskell, 16 Tennyson Rise, East Grimstead, West Sussex RH19 1SQ. Tel: 0342 312649

T A P R O U B L E

The new WP program *Perfection* was advertised in the December issue. Those readers who have good spectacles will have noted in Digital Precision's advert that the program is full of features – far too many to be commented upon adequately here. Having been using a 'beta test' version of the program, I can confirm that it is good. It will take time to get familiar with the many functions, but users who just want a faster *Quill* should not be deterred by this, as the program can be used without a lengthy study session being needed before firing it up. To try and satisfy what one might categorise as *QL World* and *Quanta* users, the user interface comes in two forms: one with menus similar to those in *Quill*, and the other having combination keying more akin to what an editor has. As many users get along quite happily without having their spelling checked, the spell-checker is an optional extra.

Perfection

Although *Perfection* is in the same market area as *text⁸⁷*, it has a different feel to it and the two programs are actually somewhat complementary. *Perfection* does not set out to make the screen look like the printed page, but it has a more obvious bias towards the programmer than *text⁸⁷* has, in that many more commands are built-in for tailoring the behaviour of the program to the user's requirements. For example, the 'lazy screen' feature can be switched on or off. This means that text can be scrolled line-by-line, with all lines being shown at all times in their correct relative positions, or the whole display except the top or bottom line can be held static while just the one line changes (very rapidly). Not surprisingly, users of *The Editor* will find much to commend in the new program. Speed has been a target throughout the development of the program, and that aim has largely been achieved, with almost every function being performed briskly. As lack of speed is a major drawback of *Quill*, *Perfection* should prove a worthy successor. Having been assured that the version I am using is about to be replaced by a much faster one, I look forward to the latter being quite something.

Adding the new program to the system disk reminded me how long it has been since I had to make any significant change to the system. It's nice not to have to keep chopping and changing, but a change

Bryan takes a first look at DP's new word processor

every now and then is healthy; it prompts you to think afresh and, maybe, do other jobs which had been forgotten. Despite the wonders of both *Perfection* and *text⁸⁷*, I still feel happier going back to *The Editor* when making changes to SuperBasic programs. You never quite seem to find one program that does everything for you, but give the new program consideration for text, database and SB programming use. Its high basic speed fits it for all three requirements.

Di-Ren, the producers of *Fleet Tactical Command*, are emphasising their hardware activities these days, and have introduced a small, cheap box for checking the health of QL networks. Called the 'network prover', this device sells for £3.50 (plus post and packing, and VAT). It is plugged in-line with the network lead between two QLs. A led on the box flashes when transmission is taking place over the network; when one QL is expecting data from another, but is not receiving it, the led comes on continuously. It was no surprise to find the network between my two QLs in a good mood when I came to test the device, but it was easy enough to simulate a fault condition by pulling one end of the network lead out, whereupon the led duly came on continuously.

Don't expect the led to flash all the time when the network is in use; if drives are running, data transmission is held up and the led won't flash, so there are only brief periods of flashing in-between long periods off, if you make copies between microdrives. The led does not come on continuously to indicate faults which are not the responsibility of the network; for example, pulling out a cartridge part-way through a copy operation is not a network fault, although it certainly halts any network operation that was in progress.

The cartridge used for checking the network prover was picked at random from a box-full of cartridges, none of which get any significant amount of use. The WCOPY command was used to copy files from flp1 on one system to mdv1 on the other. After much running of the cartridge, it became clear that something was amiss. It was not a network fault, so the led didn't come on. Being occupied with other things, I didn't check the QL for some minutes and, when I did, the fault turned out to be

a defective cartridge. The tape was jammed, and my usual treatment – bouncing the cartridge off the workbench a few times – didn't relieve the situation. Every time I have to go back to using cartridges, I get trouble. Do other disk users find this? How did we ever get any work done in the 'good old days'?

As I've mentioned before, my personal experience of hard disk drives (on a PC, I hasten to add – not on the QL), leads me to feel floppies are a more reliable medium than either cartridges or hard disks (*hear jolly well hear – Editor*). Nevertheless, there's no way I would go back to using floppies as my main storage device, after a few years using hard disks, any more than a floppy user would go back to microdrive cartridges.

Expands

Computer equipment gradually expands to fill your available workspace, and one of the things you may need at some point is a switch for your printer(s). Maybe you have two computers using the same printer, or one computer using two printers; the problems is the same, but the necessary hardware may be slightly different. Before posting your mail-order cheque off, do a bit of 'thinking on paper'. Printer switches tend to be expensive, but you can get them for a reasonable figure. A 2-way, 36-line manual switch need cost no more than about £15; paying two or three times that much will very likely get you the identical switch to the cheaper one. (I know, since I've recently spent about £30 on one, because the supplier advertising the same switch at £15 had no stock, and I was in a hurry.) A 2-way switch will have three connectors on the rear, and the connector types vary with the purpose the switch is to be put to. Check the cables you already have, between computer(s) and printer(s). Do they have 9- or 25-pin "D" connectors at the computer end, and the larger, 36-pin type at the printer end? A printer with a parallel interface will normally have the 36-pin type, male connector; they are easily recognisable, usually being blue in the area of the connections, and the connections themselves being strips of metal rather than pins. Some printers have both serial and parallel interfaces, so you may find a 25-pin connector also. In general, it is simpler to use the parallel interface, because you as a user do not have to do any setting-up for it; you simply connect the

SHOOTER

M S O L V E D

cable, whereas you may have to do some configuring (setting hand-shaking, parity etc) when a serial interface is used.

The computer is most likely to have the 25-pin D type, unless it happens to be a QL, in which case it has the BT-like PCC connector, or a 9-pin D (if it is a German or US machine). Most users will have a serial-parallel interface as a separate device for their QLs, the Miracle interface being the common one. This being the case, you don't need to worry about the non-standard connector on the QL, when connecting the QL to a switch; the connector of concern is at the other end of the interface, and is the 36-pin printer type. Likewise if you have another computer, of a different type. Whatever the connector for its printer port, you should have a cable for it which has a 36-pin connector at the printer end. If you have two computers, then, of whatever type, the chances are that the connectors on their printer cables, which will have to fit to a 2-way switch, will both be of the 36-pin parallel printer type. The remaining connector on the switch is the one which will be used for the printer. If that connector is the 36-pin type, you are going to need a 36-to-36-pin cable between it and the printer, and that is not what suppliers will generally understand if you ask for 'a printer cable'. Specify just what connectors you want on each end of any cable ordered – both the number of pins and the connector type, and whether they are male or female orientation. If you already have the necessary cables, supply the same information when ordering the 2-way switch.

When you have printer problems of the missing-characters variety, always blame the printer connections in the first instance. With so many points of contact – especially when a switch is used – the chance of a bad connection is quite high. Separate all connectors, spray the pins with **contact spray** (*definitely not* any type of lubricating oil), and reconnect, making use of any locking devices on the connectors (try elastic bands to hold them together, if necessary). If the rotary connection wafers on a switch are accessible, spray the tracks on them, too.

The information which has been quoted in another magazine, that the President of Microsoft (who produce MS-DOS) is the 33rd-richest man in the world. So far as I am aware, the source of his wealth is the one company, and basically that one program (although the WP program *Word*, the

multi-tasking program *Windows*, and a few other programs also sell well). Sales of MS-DOS have been quoted at about 11 million copies per year, which is nice money at maybe £30 average per copy (most are sold to dealers). Incidentally, Digital Research, the producers of CP/M, sell a 'compatible' replacement for MS-DOS, called DR-DOS. This is claimed to do all the things the original fails to do, such as giving access to hard disks larger than 32 MB, making 50 KB or so more space available for programs, and supporting expanded memory. MS-DOS 4.01 does provide some of these facilities now. *PC Conqueror* users might think about DR-DOS, if they haven't already got either MS-DOS or PC-DOS.

Readers' Letters

As recorded previously, my recent mail has not included complaints about the *Minerva* roms, but obviously some suppliers have been getting regular complaints. The target of the complaints is sometimes the supplier of *Minerva* (QView) and sometimes the supplier of software which has problems when run with *Minerva*. Most users will not be in a position to know whether software or hardware is causing the problem, although that does not seem to stop some of the taking a strong critical stand, one way or the other. One of the aims of the developers of the roms has been to remove known bugs in Qdos, the QL operating system. A laudable aim, but perhaps coming rather late in the day, since software writers have for many years either found ways around the major bugs or added their own fixes into their programs (or extensions files supplied with them).

Fixing known (and agreed) bugs in Qdos is one thing, but accusing software writers of having bugs in their programs is another, and one which undoubtedly raises the temperature quite a bit. This looks a bit like the situation with certain programs, where the proponents of *Minerva* are stating that they expect to see changes made to programs which have been sold for several years, and which have worked satisfactorily until the advent of (some versions of) *Minerva*. Some changes have been made, where it has been accepted that they were either necessary (regardless of *Minerva*), or could be implemented quickly and easily, to keep *Minerva* users happy. The program writers are not making other changes because they do not

accept that the problems are caused by their code; that is, they feel the fault lies in *Minerva*. A little more consultation before some of the rom development took place might have prevented all the users' problems.

Here are some notes on reported problems. **Some** versions of *Minerva* don't get along with **some** of the toolkit functions with the *Turbo* or *QLiberator* compilers, in **some** QLs. **Some** versions of *Minerva* and *Professional Publisher* don't work too well together; specifically, the 'Edit Page' close-up window can be blank, and high-definition fonts may be unusable. Version 1.81 *Minerva* prevents *Microbridge* loading fully; this may be solved by increasing the Dataspace allocated to *Microbridge* by 1 KB (if you have the *Turbo* dataspace routine available). Unspecified programs will not work unless the *Minerva* integer tokenisation function is switched off. *FlashBack SE* displays minor quirks when used with *Minerva* V1.81; *Touch Typist* does not load fully with the same version. The *SEXEC* command gives loading problems with some programs, when V1.82 *Minerva* is used.

Spencer Harvey reports a lack of response (as of mid-November) from **Sector Software** concerning a printer interface, returned by him as faulty in June 1990. It has been suggested that the interface never got back to Sector, but letters concerning it did, and it would seem reasonable to have had some reply to them by now. A QL user who seems to have regular problems with his equipment (hardware and software) is **J Roy Goodall**, and his location (Belize) makes sorting-out problems a rather difficult and protracted business. He eventually received a long-delayed printer interface from **Miracle Systems**, along with a cheque which was apparently a refund of the cost for it, and has also purchased a Trump Card. He tried out *TechniQL* on a QL he has recently obtained, but had 'the usual negative results'. Another acquisition is a PS/2-style keyboard, from **Keyboard Products**; neither of his QLs will give the F1/F2 start-up screen with this keyboard fitted, so he is now awaiting comment on this problem. He says his rechargeable batteries for the **Frequency Precision** power back-up unit are in need of replacement, but this device has seen him through "many blackouts, and also overcame overheating problems", over a period of two years.

THE ITALIAN JOB



The QL is still active in Italy, where the annual QL Users Meeting 1990 took place recently by Lake Garda. The verdict: a success, but rather hard to get there.

This is a user's-eye report from the Second Italian QL Users Meeting which was held at the Management Conference Centre at San Felice del Bannaco which is on the southern shores of Lake Garda, halfway between Milan and Venice.

The southern shores of Lake Garda are a beautiful part of the world to spend your time if you have any but there was no time for sightseeing at the Meeting.

Committee

The Meeting was as far as the Brits were concerned ably organised by Eros Ferrenzi, who is a student. The Users committee consisted of industrialists and enthusiasts from northern Italy and the company called SPEM, owned by Guido Masoero, who operates from Turin and who

the principal QL trader in Italy.

Miracle Systems, who usually travel to QL meetings by bicycle, left their bicycles behind and travelled by car to Dover and then to Brescia by train from Calais. Bill Richardson of EEC Ltd chose to fly carrying several kilos of QL product in his baggage, but despite Al Italia hospitality onboard found himself incarcerated in the Customs sheds at Milan Airport for three hours, missing the opening stages of the afternoon, when not many people attended. Another trade attendee was Jochen Merz from Germany, showing an updated Version 3 QD Editor and an improved range of software.

Sunday was a beautiful morning and it would have been nice to have gone sightseeing rather than the Show but on arrival we found twenty

to thirty QL enthusiasts waiting for us, mostly from local areas but several had travelled from as far as Rome and San Remo on the French/Italian border.

Audiovisual

During the course of the day several QL presentations were held in an excellent lecture hall well equipped with audio visual paraphernalia and material put over by the Italian delegates, Miracle Systems' Stuart Honeyball and Jochen Merz. All was efficiently translated into Italian by the members of the User Group. The opinion of the UK exhibitors, Stuart Honeyball of Miracle Systems and Bill Richardson of EEC Ltd was that, while sales were good and most of the stock taken by both companies was sold, sales takings were great in millions of Lire but condensed into

pounds sterling it probably represented the kind of sales that one would have expected from a one day Show at the Horticultural Hall in London. Expenses were well exceeded, though, and it was felt that a number of orders will follow the visit. Quanta leaflets and back numbers on the EEC stand attracted a lot of attention and a number of new members will result.

Only about 100 visitors attended however and all exhibitors felt that the same faces seem to be around who we had previously met and it was thought that, bearing in mind the somewhat remote tourist area around Lake Garda, it would be nice to see other shows in the more populous areas of Italy such as for example, Turin or Rome and it was suggested by an attendee that a good place for a QL meeting be Florence in future.



SUPER BASIC

Most computer users occasionally write code to start software. Mike Lloyd provides a step-by-step guide.

Every computer system needs them, yet many computer users cannot create them or modify them. Some are remarkably simple, while others are unavoidably complex. Most of them are essential, but on occasion they are merely convenient. Some have a single purpose, others offer a wide choice of options. Some never change, but many need to be modified to suit particular needs. They are, of course, boot files: short programs required to load and execute other pieces of software.

Most of the requests I receive for help relate to boot files, particularly ill-considered amendments to them which need to be put right. A quick straw poll amongst software sellers and producers reveals that boot files come pretty high up their list of reasons for customer calls. It is not that the boot files supplied with the software cause problems; it is the enthusiastic but badly-planned adjustments to standard boot files which lead to problems.

Bootfile

This SuperBasic article and its successor are devoted to the techniques and principles of the humble boot file, including a complete review of the commands involved, a look at boot file editing methods, some hints and tips, and a simple but fully-functioning menu-driven boot file which can be adapted to suit your own software library. I hope it answers the vast majority of questions about the subject and reduces the number of calls for help to suppliers.

To simplify the text and to avoid saying everything twice, I have assumed that floppy disks are being used throughout the rest of the article. If you are still using microdrives, replace all occurrences of FLP1_ with MDV1_ and translate all mentions of 'floppy disk' and 'disk drives' to 'microdrive cartridge' and 'microdrive'.

A boot file contains a number of program commands which are carried out

when a disk or microdrive is accessed during the boot-up sequence when the computer is first started. This is incredibly useful because, instead of being faced by a blank screen and a flashing cursor, users can be presented with their favourite program loaded and running automatically.

Disk User

The main purpose of a boot file is to issue the commands needed to load and execute another, larger program. It might also contain some commands to change aspects of the QL's settings to meet the needs of the software it will launch. Disk drive users, who can place many substantial programs on a single disk, often develop a boot file with a menu so that they can select which software they wish to run. This is less likely to be the case with microdrive users because it is often not possible to place more than one large program on a single microdrive.

There is nothing magic about a boot file's contents. Usually, they contain the commands which would otherwise need to be typed in by the user directly whenever a piece of software was to be loaded. Generally, the commands in a boot file fall into three categories: file-loading commands, environment-setting commands and incidental commands. The file loading commands perform the job of loading programs from disk or microdrive and starting them off. Environment-setting commands might modify the screen settings, or establish parameters for the printer channel, so that the main program knows exactly how the QL will behave. Some users add extra commands to set up ram-disks, or make multi-tasking software available, in order to meet their own particular needs. Incidental commands include special screen displays, or perhaps a menu.

Likewise, there is nothing very special about the file itself. It is simply a Qdos

convention that when a microdrive is present in mdv1_, or a floppy disk is in flp1_, during the start-up sequence the operating system looks for a file called 'boot'. If it does not exist, nothing happens, but if it does it is assumed to contain SuperBasic commands which are executed. If you forget to put a microdrive or disk in place when starting up the computer, type LRUN FLP1 BOOT in the command window and press Return. LRUN is a SuperBasic command which means "Load and RUN" and can be used to launch any SuperBasic program.

Incidentally, boot files are not particular to the QL: all modern operating systems have them. In MS-DOS, they are called AUTOEXEC. BAT files and in Unix they are called .PROFILE files, but their purpose is the same as the QDOS BOOT file. Of the three, the Qdos name is by far the most meaningful.

Boot files are always standard SuperBasic program files, rather than being machine code files. This ensures that they are easily readable and adjustable. To load a boot file without executing the commands it contains, start up the QL without a disk or microdrive in place. The QL screen will be set to the default according to whether the start-up sequence was initiated by an F1 or F2 keypress. The screen will contain three windows, although in the tv display mode only two are immediately apparent. The lower window is the command window, where SuperBasic commands are typed and program lines are modified. Above the command window are the listing window and the default window. In Monitor mode these windows are side by side whereas in TV mode the default window is superimposed over the listing window.

To view a boot file, place the appropriate disk in the flp1_ drive and type:

```
COPY FLP1_ BOOT TO SCR_
```

The boot file will then be displayed in the

centre of the screen. Press the F5 key to pause scrolling if the beginning of a long file looks in danger of scrolling off the screen. If each line begins with a number then the file can be loaded directly into memory. If there are no leading line numbers we must add some before we are able to edit the boot file.

To load a numbered boot file, place its disk in the appropriate drive and type in the command to load it into the QL's memory:

```
LOAD FLP1_ BOOT
```

The disk drive will spin into life and if a file called BOOT exists it will be loaded and the cursor will flash ready for the next command. To view the file's contents, type:

```
LIST
```

Sometimes, software developers write boot files which do not have line numbers, for reasons best known to themselves. If you wish to edit such a file, you can copy the details of the file from the screen (or a printout) and type them in again with line numbers, or you can use the short routine at **Listing one** to do the job automatically. The file it creates is called "NEWBOOT". When this file has been edited, delete the old BOOT file and save the amended version as "BOOT".

Before we can begin to modify these commands their purpose must be understood. The heart of any boot file comprises the instructions to run the main program or programs, so let us look at those first.

WARNING: Always edit a copy of the original boot file, not the original itself. That way, if you make a mistake it can be rectified by taking a new copy of the original boot file and starting again.

There are three ways of causing a program to be run from file on the QL: the LRUN command for SuperBasic programs, the RESPR/ LBYTES/CALL sequence of commands for machine code programs and the EXEC family of commands for multi-tasking programs. Variations of these commands are avail-

able in toolkits, and will be covered at the appropriate points below.

It is unusual for SuperBasic programs to be launched from boot files, but it can be convenient to do so and therefore the method is detailed here. The command is LRUN followed by the file name. Note that there is no point in following the LRUN command by any other commands as they will never be carried out: the incoming SuperBasic program completely overwrites the commands loaded from the boot file.

Very little software for the QL has been produced in conventional machine-code format because it is easier and more advantageous to make such programs multi-tasking. However, it is very common to see machine-code format programs which set up a multi-tasking program of some sort. Perhaps the prime example is Qram, the Multi-tasking utilities package. It is machine code programs such as these which lead to the majority of problems encountered by *QL World* readers.

Three Stages

With standard Qdos and SuperBasic, loading a machine code program is a three-stage process. An area of the QL's random access memory (ram) must be specially designated for the program, the code must be copied from the disk into the designated memory area and, finally, the code must be started off.

Memory is designated using the RESPR (REServed PProgram space) function. You ask the computer to reserve a memory area of so many bytes and the computer in turn tells you where that memory areas starts. The syntax for the command is:

```
LET base = RESPR (2048)
```

Here, the request is for a 2K chunk of memory and the first byte in that chunk is at the address number which Qdos will have placed in the "base" variable. You could just have easily typed:

```
PRINT RESPR (2048)
```

which would have reserved a space and printed its first address on the screen.

Putting the address into a variable saves the effort of writing it down and typing it directly into subsequent commands.

You can find out how much space to reserve for a particular program by referring to the product's manual or by using the WSTAT Super Toolkit command. Memory blocks are allocated in lumps of 512 bytes, so there is no requirement to reserve exactly the right amount of space, provided that there is enough roped-off memory available. If you intend to load two or more machine code programs sequentially, you can reserve a single space to contain them all, provided that each is loaded beginning at an address which ensures that none are overwritten.

RESPR will fail if there is insufficient ram available, or if there is a multi-tasking program running when it is called. The first restriction can be solved by removing any other reserved memory blocks which might be present, or by buying extra memory for your QL. The second restriction has been overcome by replacing the RESPR command with toolkit functions such as ALLOCATION, from the Turbo toolkit.

Having reserved space in memory, the machine code program must be loaded into it. The command for this is LBYTES, followed by the file name and the address of the first byte of reserved memory space. To follow on from the RESPR command above, a typical command might be:

```
LBYTES flp1_ little_ prog, base
```

If a single ram area has been reserved for a number of machine code files, careful control must be maintained of the starting

Sharing

address for each LBYTES command if problems of over-writing are to be avoided. Additionally, it is a Qdos rule that machine code programs must begin at an even-numbered address in memory. Let us take as an example the following Qram files, with their approximate sizes:

```
flp1_ wman (5006 bytes)
flp1_ ptr imi (9654 bytes)
flp1_ ramprt (1024 bytes)
```

Memory space can be reserved and occupied by the following commands:

```
LET progA = RESPR (15684)
LBYTES flp1_ wman, progA
```

```
LET progB = prog A + 5006
LBYTES flp1_ ptr_ imi, progB
```

```
LET progC = progB + 9654
LBYTES flp1_ ramprt
```

The final part of the sequence is a CALL command which activates the code loaded

LISTING 1

Note: this listing is a stand-alone program which should be saved (eg SAVE flp1_editboot) before being run. Do not type the italic remarks.

```
100 OPEN_IN#3, FLP1_BOOT
110 OPEN_NEW#4, FLP1_NEWBOOT
120 LET NUMBER = 10
130 REPEAT LOOP
140 IF EOF(#3): EXIT LOOP
150 INPUT#3, A$
160 IF A$(1) <= "9": EXIT LOOP
170 PRINT#4, NUMBER & " " & A$
180 NUMBER = NUMBER + 10
190 END REPEAT LOOP
200 CLOSE#3
210 CLOSE#4
220 PRINT "COMPLETED"
```

```
Open existing boot file
Open a new file
Set line number value
LOOP:-
Stop if file end reached
Read next line from BOOT
Stop if its already numbered
Add numbered line to NEWBOOT
Increment the line number
END LOOP
Close the BOOT file
Close the NEWBOOT file
Signal end of program
```


SUPERBASIC

by the previous commands. The CALL command needs one parameter, the address of the first byte of executable code. Strangely, this might not always be the very first byte of the reserved area, in which case the software manual will advise you to use a command such as:

CALL base + 12

Fortunately, for everyone who finds this three-command sequence a bit of a bind, Tony Tebby's *Super Toolkit 2* contains a single command which performs all three tasks and which does not require the user to worry about start addresses, even-numbered bytes and 512-byte memory chunks. The command is LRESPR, and it is followed by a single parameter — the program's file name. Therefore, instead of typing:

```
LET base = RESPR (1024)
LBYTES flp1_ program, base
CALL base
```

the following single command can be used instead:

LRESPR flp1_ program

The final category of runnable files comprises those that can be multi-tasked together. To take full advantage of multi-tasking software it is best to purchase

Qram or Qpac II because these products modify Qdos to improve the multi-tasking environment. Almost inevitably, if you intend to load more than one substantial executable program at a time your QL will need memory expansion.

The commands to load and initiate an executable program are quite straightforward. If the program is to work at the same time as other multi-tasking programs the command is:

EXEC flp1_ filename

If a number of programs are launched in this way they will share processing time and appear to be working independently and simultaneously. For programs which need keyboard input this can be a problem — which of the programs should respond to a particular set of keypresses? Which program is going to have first access to the printer, or the disk drive? Qdos has no simple answers to these questions, and so its multi-tasking environment needs to be tamed.

The easiest way of preventing input and output clashes is to tell Qdos that the program being run takes full control of all device channels and processing time until it is finished. The command to do this is EXEC_W, and its syntax is exactly the same as for its EXEC_ relative. The W extension means "Wait until terminated". For the vast majority of executable pro-

grams commercially available the EXEC_W keyword should be used as a matter of course. It is nevertheless worthwhile to be aware that under Qram and Qpac II the QL's multi-tasking environment is changed so that EXEC and EXEC_W have the same effect.

Super Toolkit 2 expands the EXEC family to include two more keywords, but their full potential is of more benefit to knowledgeable programmers than for non-programmers. However, for convenience use can be made of:

EX Exactly the same as EXEC
EW Exactly the same as EXEC_W

Turbo Toolkit provides EXECUTE, EXECUTE_A and EXECUTE_W as other alternatives. These commands all allow more parameters than just a filename, but their use lies beyond this article.

There are advantages which can be gained from the features of a multi-tasking program without necessarily running it simultaneously with another program. The Psion business utilities bundled with the QL are all multi-tasking executable files which can all fit on a single disk, but only one of which can be loaded into a standard QL ram at one time. Nevertheless, a form of task-switching can take place provided that an appropriate boot file is constructed. The details of this boot file will be printed next month, along with many more useful and practical hints about boot files.

DI REN

Fleet Tactical Command II Di-Ren's Software Flagship Version 1 was unique and brilliant. Version 2 is in a class of its own!!

Version 2 of this unique, Real Time, Fully interactive, 3-D Naval Warfare scenario simulator, that can be played via two computers connected via Network or Modem links, has many new features.

'Weather conditions' for instance can range from 'calm' to a Force 11 Violent Storm! The Crew suffer fatalities and you can now 'Tow' ships. A most significant advance is the 'TWO PLAYERS ON ONE MACHINE' option. This option is invoked by a 'Swap' command and requires an expanded QL to run. Even now however Networking and Single User options can still be run on an unexpanded QL.

To describe FTC II in full would take pages, so write for your free Handbook - 'A Look at Fleet Tactical Command II'.

Price: 3.5" Disk Version - £49.95 2 Masters supplied
MDV Version - £56.95 4 Mdv's supplied

SPECIAL OFFER

Fleet Tactical Command V1.02

A limited number of Version 1 copies are available on a 'First come - First Served' basis. Customers taking advantage of this offer can upgrade to Version 2 at a cost of £20.00.

Special Price: 3.5" Disk - £29.95
MDV - £32.95

This special offer is only available direct from DI-REN (UK). All FTC Packages include charts, scenario logs and geometry sets etc.

The Fugitive

Not flashy, easy or friendly is an excellent description for this Text Adventure. You are a fugitive in Russia and have to leave in a Hurry!

This will keep you occupied for weeks.
Price: £9.95

Mini Process Controller

The MPC is a remarkable VERY LOW COST switchgear unit that can be plugged into ANY computer's Centronics Interface (as commonly used on printers) and allows the user via very simple programming techniques to operate 6 make/break relays.

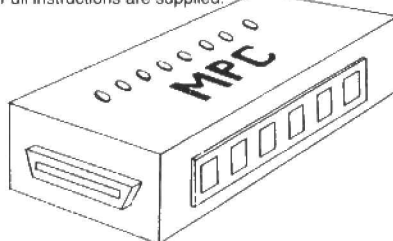
This unit is of interest to hobbyists and professional users alike. Switch timings can range from 24 hour to split seconds. Its uses are numerous, from energy management to security switching and even switching off the portable TV!

A major feature of the unit is the ability to connect it permanently in-line with the printer. The printer can be used without affecting the controller.

For ease of compatibility sockets are fitted to the unit that any 'Figure of 8' plug/lead (usually found on cassette recorders and so on) can be plugged in to. These are easily obtainable. One is supplied with the unit.

The unit can be powered by 2*PP3 (or equivalent) batteries or by an optional UK type mains adaptor.

Full instructions are supplied.



Prices:
Mini Process Controller (Batteries no included) £49.95
Spare connector leads £0.85
1 Metre printer extension cable £7.95
2 Metre printer extension cable £9.95
UK Type Mains adaptor £8.95
QL MPC Utility £9.95

QL MPC Utility programme

This menu driven programme sets up and installs critical and long-term timing operations for the MPC. It can Save/Load data files and contains some useful Superbasic commands.

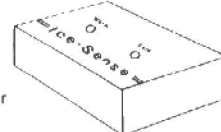
A Small non-destructible on-screen display indicates the controller state.

Di-Ren Network Prover

This simple but invaluable device is essential for those of you who regularly use the QL-QL network. Plugging it in between the connected QL's indicates, using an LED, whether transmission is taking place or if the network has 'sleazed up' thus keeping you informed of what is actually happening. The price? just £3.50.

Ice-Sense

As a special offer to QL World readers our automotive product 'ICE SENSE' is yours for only £4.95



This electronic 'Black Box' product is easily fitted to most vehicles with a 12 Volt supply and gives warning of impending ICY ROAD conditions.

Availability

DI-REN products are available from:

Di-Ren
43 Davids Road
London
SE23-3EP
Tel 081 291 3751

QLympic Computer Systems
Quellenweg 18
4220 Dinslaken
W.GERMANY
02134/91766

Ordering From DI-REN

For Software orders state Disk or MDV.
All prices inclusive of P&P (UK, Europe/EEC)
Access/Visa telephone orders accepted.
Cheques should be made payable to DI-REN.
Clearly state your name, address and product required.
Further product details available on request.

SOFTWARE FILE

DOUBLE BLOCK

Rich Mellor goes Russian in an old game brought up to date.

This game is based upon a popular Russian game called 'Tetris' which is in itself a simple, highly addictive game. The idea of the game is that several different-shaped multi-coloured blocks fall from the top of a 'pit' on screen. These must be rotated and moved left or right so that they form a full line of blocks across the screen. Once a line has been completed, it disappears from the screen so that more lines may be formed within the 'pit'. Should any line have a gap (or two) in it, this will remain on screen, with the pit gradually filling up with incomplete lines. Once there is no more room in the pit to allow another block to be dropped, the game is lost.

With the idea behind the game being so simple, the keys must also be simple. The left and right cursor keys move the falling blocks left or right, the up cursor rotates the block, and the down cursor makes the block drop more quickly. This is a straightforward combination which allows a joystick to be used if necessary. However, this version of the game also incorporates a two-player mode, where both players have a pit on screen, each with its own set of falling blocks. One player can use the cursor keys, while the second player really needs a joystick plugged into CTL2 (although the function keys F1 to F4 can also be used).

Colour

The display on screen is well laid out with lots of colourful graphics, and even a little sound to keep you amused. Also shown on screen is your present score. There is an option in the one-player mode to

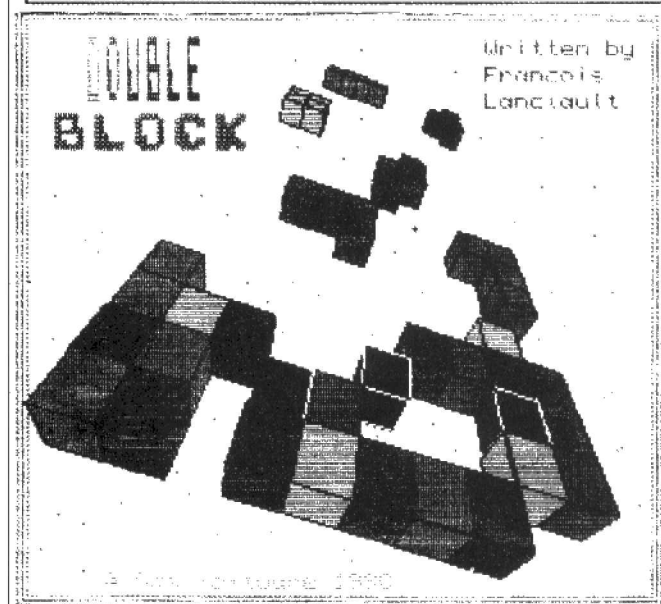
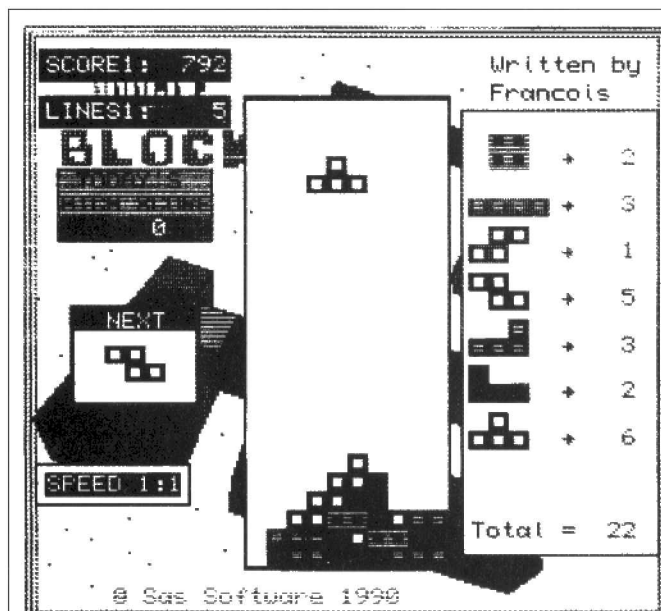
show the next block which will be dropped, so that you can plan ahead. Peculiar to this version of the game is an extended graphics mode, where the blocks can be any one of 16 shapes (the normal set features only seven). This extended graphic mode certainly makes the game much more difficult and should prove a challenge to the most practised Tetris player.

Points are scored in this game for each block which falls and each line which is formed. The game maintains a high score table, which on start-up contains some high scores achieved by the author: these are depressingly high however, and you will be glad to learn that you can get rid of the current high scores at any time merely by deleting one file from your disk.

Faster

The game also features 10 different speed levels; these can be different, if you are playing in two-player mode, for each player. However as you play the game, the speed of the falling blocks increases, so that with the screen slowly filling with incomplete lines, you soon begin to realise that your reaction times are much too slow. It was however nice to see that the speed increase is so set that in the two-player mode, if one player begins at speed 0 and the other at speed 6, you will both reach speed 7 at the same time! Still, if it all gets too much for you, a press of the 'P' key will pause the game while you get your breath back and decide just what to do with the mess you have made on screen.

In all the game is a most welcome rest from the normal type of arcade game which appears all too readily on home com-



puters. This game requires thought, quick reflexes and an eye for opportunity. It reminds me of that old children's educational toy where they have to fit the different shapes into the different holes, the only difference being that here much more speed is needed, and a better eye for patterns.

INFORMATION

Product: Double Block

Price: £9.00

Supplier: CGH Services,
Cwm Gwen Hall, Pencader,
Dyfed, Gwyn SA39 9HA.
Tel: 0559-384574.

Technical Helpline

Novice

I have just acquired an old QL, and as a complete novice to programming have been attempting to write some software to make my business run more smoothly. There are however a few things that I have been unable to find out how to do.

**Mark Smith
New Costessey
Norwich**

A) How is it possible to have a key that can over-ride the program at any stage during the run, (same principle as the ESC key in Quill)?

Pressing <CTRL & Spacebar> at the same time breaks a SuperBasic program at the end of the current statement, giving the error message At line NNN not complete. As with any error message there are then two commands you can type to get it going again: CONTINUE carries on just where the program left off while RETRY goes back and does the last statement again before continuing — this can be useful, as when you realise you have the wrong mdv cartridge in the drive and want time to change it. You can sometimes actually EDIT a line and then RETRY, but this rarely works properly; it certainly won't work if you are in the middle of a PROCedure or FuNction, or any kind of loop; the program tends to forget where it was in these when you EDIT.

B) Is it possible in one line to check that an input is numeric? (In order to stop a program crashing if a non-numeric amount is entered in a numeric variable.)

A SuperBasic extension to do this, CHECKF, was described by Simon Goodwin in *DIY Toolkit* in the January 1989 issue of *QL World*. This is now available on disk or mdv for a small fee. SuperBasic function can be written to do the same job, but it takes about 35 lines.

C) How is it possible to copyright a program (to stop other people being able to list it)?

What I think you mean is to copy-protect a program copy-right refers to the legality rather than the feasibility of making copies. There is not much you can do to prevent others from reading your SuperBasic listing, unless you compile it — there are three compilers on the market, and that does make it unlistable, but the compiled versions can still be copied. There are a number of tricks to make copying more difficult, but, particularly with disks a determined hacker can defeat them. However, since you admit you are a beginner why would you want to do this? It is not very likely that you will immediately be able to write

the ultimate QL program, and make your fortune! Fortunately many competent programmers are prepared to rely on the law of copyright and allow users access to their code. That is one of the ways a beginner can learn: study other people's programs from wherever you can get them — MDX exchange, Quanta library if you are a member (and any serious QL user should be), QL World listings, etc. and see how they do it. Once you find yourself saying, "I can see a better way to do that," you are on your way.

Background

Please can you tell me the command which changes which part of the ram the screen is, also can you tell me how games writers stop the background graphics being erased when the sprites walk over them? I am hoping to include these in a Basic program.

**Peter Smith
Middlesbrough
Cleveland**

Although Sinclair planned a

Sprite Listing

```
10 CLS: FOR i=0 TO 208: PRINT "Fred";
15 OVER(-1): REMark XORs screen output
20 FOR i=0 TO 200
25   Sprite i+5,50+50*SIN(PI*(i)/50)
30 END FOR i
35 FOR i=205 TO 5 STEP -2: Sprite i,50
40 OVER 0: REMark Normal printing again
45 REMark ~~~~~
50 DEFine PROCedure Sprite(x,y)
55   LQCal c: c=7: REMark colour of sprite
60   BLOCK 2,10,x,y,c: BLOCK 10,2,x-4,y+4,c
65   PAUSE(1): REMark To adjust speed
70   BLOCK 2,10,x,y,c: BLOCK 10,2,x-4,y+4,c
75 END DEFine
```

Howard Clase picks up some readers questions about technical aspects of the QL.

two screen mode for the QL he never implemented it in any of his versions of the rom. However QView's *Minerva* rom does have this facility along with many other improvements.

True sprites require considerable expertise in writing machine code, but you can make use of the OVER -1 command in SuperBasic to get a similar effect. This has the effect of XORing any printing or graphics in a window against the background until the OVER command is used again. So that printing the same thing twice at exactly the same spot causes it to appear and disappear again, restoring the original background. Because it is in SuperBasic it won't be very fast. The short program in the listing illustrates the principle.

The 'sprite' here is a simple cross which moves in a sinuous path over the text, returns straight to the starting point and then vanishes. The more complex your sprite the slower will be the movement. I've drawn the sprite twice within the PROC; this is faster than calling it twice from within the loop. It is best to design your sprite with the BLOCK command if you can rather than with the graphics commands because they are also slower. Changing the parameter of the PAUSE command in line 65 can be used to adjust the time the sprite spends on the screen, for maximum speed remove it completely. Use the variable c in the procedure to experiment with different colours.

Rounding Up

A German reader, Wilhelm Albrecht, has noticed that the

Listing 1

```

100 CLS
110 n=4.5*18.05
120 PRINT Round(2,n)
130 STOP
140 REMark ~~~~~
150 DEFine FuNction Round(p,n)
160 REMark p = No. of decimal places
170 LOCAL m
180 m = INT(n*10^(p+1)+.5)/10^(p+1)
190 RETURN INT(m*10^p+.5)/10^p
200 END DEFine

```

standard 'rounding off' routine behaves erratically when the digit being dropped is exactly 5. The SuperBasic formula for rounding a number N to p decimal places is: $\text{INT}(N \cdot 10^p + 0.5) / 10^p$, and Mr Albrecht has discovered that when $N = 4.5^{*}18.05$ rounding to two places the result (81.225) is rounded down to 81.22 instead of up to 81.23 as he would like. He

suggests adding 0.5001 instead of 0.5, but while this corrects this particular problem it would create new ones, rounding up numbers like 18.7499999 when it ought to round them down.

Like Mr Albrecht I was taught to always round up under these circumstances, but here in North America the convention — and that is all it is — is to round a 5 to an even

number. The argument is that this will randomly result in an equal number of up and down roundings, so that the total of a large number of rounded numbers will be more accurate than the total of a set produced by the European method.

Binary

The QL, like all digital computers, does all its calculations in binary, converting decimal numbers back and forth as required; often there isn't an exact conversion, but Qdos does live up to Sir Clive's promise of seven digit accuracy, and Psion have been able to build twice that into their programs. Mr Albrecht's problem arises from this binary to decimal conversion error. The actual result of the QL's calculation of $4.5^{*}18.05$ is 81.22499994039536 which

Qdos correctly prints as 81.225 to seven digit precision. Note that this is accurate to 9 figures. (How did I find this out? Try PRINT 4.5*18.05 - 81.225 and you will see the difference.) The situation is similar in Abacus except that the arithmetic is "double precision". In both cases the INT function operates on the number actually stored in the QL's memory and not the rounded value however meaningless the last 7 or 8 digits may be.

When presented with a true exact 5 to round the formula does always round up, so Mr Albrecht can get the behaviour he wants by a two step rounding: round first to three places and round the result of that to two; this still may fail very occasionally, but should be much more reliable. **Listing one** gives a SuperBasic FuNction that will do this. But, while I was writing and testing this, I discovered another problem, there is a limit to the argument of the INT function of +/-2147483647; values outside this range generate errors both in SuperBasic and Abacus. It is thus possible to cause an overflow when dealing with large numbers or when rounding to a large number of decimal places. In SuperBasic this shouldn't be a problem if you remember the normal seven digit precision, but it could be a limitation for accountants using Abacus for dealing with multi-million budgets!

Uncertainty

The real point is that there is no correct answer to this problem, and in fact Qdos's behaviour is probably better (more random) than either convention. The fact that you want to round implies that the digits 'being lost' are inaccurate or unreliable (if they originate in a measurement) or irrelevant (if they result from an exact calculation). You have to accept that some uncertainty still exists in the last figure and to minimise this you must work with at least one more significant digit than you really want, rounding only at the very end.

```

A B C D E F
1      EXAMPLE FOR NOT CORRECT ROUNDING
2
3          MULTIPLYING NUMBER 1 TIMES NUMBER 2 AND ROUNDING
4
5
6    NUMBER 1   NUMBER 2           N1 * N2               N1 * N2       INT(N1*N2
7                                     #100+.5)/100
8
9              10 FIGURES BEHIND DEC.-POINT     2 F.B.DP        5 F.B.DP
10
11
12         2.5    13.41                33.5250000000            33.53             33.52000
13         3.5    13.41                46.9350000000            46.94             46.94000
14         4.5    13.41                60.3450000000            60.35             60.35000
15         5.5    13.41                73.7550000000            73.76             73.75000
16         6.5    13.41                87.1650000000            87.17             87.17000
17         7.5    13.41                100.5750000000           100.58            100.58000
18         8.5    13.41                113.9850000000           113.99            113.99000
19         9.5    13.41                127.3950000000           127.40            127.40000
20
21         TOTAL :                      643.6800000000            643.6B             643.70000
22
23 MIND THE MISTAKES :
24 Fields D12 to D21 are correct !
25 Field E21 = 643.6B ist not the sum of D12 to D19 as described in MANUAL
26 But now look at fields F12 and F15!
27 Rounding up are not correct ! Mysterious things are happening. WY ?
28 Type in to field D29 : >D12-33.52<
29                        -0.000000000000000008881784197
30 Type in to field D31 : =>D15-73.755<
31                        -0.000000000000000017763568394
32
33 The same bug exists by using SUPER_BASIC without ABACUS !!
34 One way to go round this :
35 Using the INT= Kommand don't only add .5, but .5001 after multiplying
36 with 100,1000,10000 etc. and before deviding through 100,1000,10000.
37
38 May be there is a better way.If so,please tell it to me.
39
40 To get this sheet ,the following formulas are used:
41 Abacus GERMAN VERB 2.45 ROM VERS=MGG ENGLISH may be :
42 D12 sp=xle()-9.5 von 12 bis 19 col=line()-.9.5 from 12 to 19
43 C12 sp=13.41 von 12 bis 19 col=13.41 from 12 to 19
44 D12 sp=B12*C12 von 12 bis 19 col=B12*C12 from 12 to 19
45 E12 sp=B12*C12 von 12 to 19 col=B12*C12 from 12 to 19
46 F12 sp=GANZZAHL(B12*C12*100+.5)/100 col=INT(B12*C12*100+.5)/100
47      von 12 bis 19                  from 12 to 19
48 D21 SUMME(D12:D19)                   sum(D12:D19)
49 E21 SUMME(E12:E19)                   sum(E12:E19)
50 F21 SUMME(F12:F19)                   sum(F12:F19)
51 D29 and D31 : look above
52
53 Wide of columns
54 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

```


30 THINGS TO DO WITH A QL



In the same way that many expensive high-spec cameras are only ever used for taking holiday snaps, scores of QL computer owners are only using their superb machines to carry out a limited range of work. It is worth taking another look at some of the many tests the QL can perform – and here are thirty of them just to start the ball rolling.

PLAY GAMES – the obvious one, with plenty of software available ranging from the many Quanta freebies such as *Star Trek*, *Dungeons*, *Duckshoot* or *Mastermind* to commercials such as *Microbridge* which could set you back £35. Old favourites – but still unbeaten – are *Scrabble* and, of course, *Psion Chess*.

David Drysdale lists creative ways people have put their QLs to work. And yes, we would like to hear from readers who can add to the list.

LEARN TO TYPE – something which very few people learn to do properly yet, once mastered, this skill will pay dividends throughout life. Try *Touch Typist* from Sector Software at £12.

SET UP A STAND AT A GARDEN FETE – plenty of fund-raising scope if you have a power source to get the QL up and running. Horoscopes and biorhythm charts are two possibilities. Software can be bought but for the fun occasion try concocting something of your own.

MAKE TITLES and GRAPHIC DISPLAYS for your slide show or video production. There are programs, at every level, everywhere you look.

GET TO GRIPS WITH ARCHIVE – it really is an amazing piece of software once you get used to the procedures and Sedit screens. All will be revealed in Mike O'Reilly's book *Database Management on the QL* if you are lucky enough to find a copy. Or try the *Archive Tutor* program from TK Computerware.

PLAN YOUR HOUSEHOLD BUDGET FOR THE YEAR – there's no finer way to learn the wonders of the Abacus spreadsheet. Once your budget is up and running ask some 'What if?' questions about your personal spending. The spreadsheet's lightning recalculations may throw up some surprises.

MAKE A COMPUTERISED CHRISTMAS CARD LIST – on Archive, of course. You can bring it on screen each year and update it. Remember to record gifts sent as it is all too easy to send the same type of gift year after year automatically.

BECOME A PRINTER – the variety of desk top publishing programs offered in Sinclair QL World should enable you to print like a professional. It won't happen however without a printer's trained eye for type design. Look out for books and courses on the subject.

KEEP A PERSONAL SPENDING ACCOUNT and discover why cash disappears so fast. *Bank Account* from SD Microsystems is well worth considering.

Charity

MAKE A CARD INDEX OF YOUR SLIDES AND NEGATIVES – again on Archive. Keep plenty of fields available for searches and to reduce cross-referencing.

MAKE A GRAPHIC DISPLAY FOR A CHARITY – most charity stalls at fetes and fund raising fairs like something on the stall to attract attention. Ever changing charts and graphs appearing on your screen will do just that.

MAKE AN ADVERT DISPLAY FOR A SHOP WINDOW – if you can stand being without your QL for a week or so at sale time, a shopkeeper friend would be grateful to have the bargains displayed this way.

HELP A CLUB WITH ITS MEMBERSHIP LIST – clubs and small charities are always grateful for a computerised membership list that can give up-to-date printouts. Check with the Data Protection people before you make a start though.

MONITOR YOUR UNIT TRUSTS AND PRIVATISATION SHARES – *Investment Monitor* by Michael Slatford is ideal for

this. (TK Computerware £20). And don't forget *Easel* for share performance graphs.

DEVISE A STOCK CONTROL SYSTEM FOR YOUR FREEZER – with shopping lists and an updatable printout to fix on the freezer door with a fridge magnet.

MAKE A FAMILY MEAL PLANNER – with rotational recipes, and save the cook's constant worry about what meal to serve.

GO ONLINE TO PRESTEL – you'll need a modem of course; look out for the Tandata three stack modem which can sometimes be obtained from Quanta at an advantageous price. There are hundreds of information providers plus computer clubs, an agony aunt, an AIDS information service and a Gay line.

Prestel

BOOK A CHEAP HOLIDAY – again through Prestel. Many travel agents have cancellations and last minute bargain offers and Prestel is the ideal medium for picking them up.

OPEN A COMPUTERISED BANK ACCOUNT – you'll need the modem to see your bank statements on screen and to make instant cash transfers between deposit and current accounts. Pay all your bills from the keyboard and pick up a bank statement from your printer whenever you like. Contact: Home and Office Banking Service, Bank of Scotland, 2 Robertson Ave, Edinburgh EH11 1PZ.

TIME YOUR PHONE CALLS AND KEEP BILLS DOWN – lots of scope here for designing a program to keep track of the time of day, length of call and the cost of each unit. A discreet little beep for every 50p spent might be a good idea too. For people who are not too advanced in programming a ready made call timer can be had from Quanta. This is quite good, but screams out for further development.

MAKE A POOLS FORECASTING PROGRAM – there are plenty of basic coupon marking systems around including those issued by the pools companies to act as a basis. You can buy one readymade of course. Advertised occasionally in QL World.

BECOME A RADIO OPERATOR – wireless telegraphy using morse code has not been totally eclipsed by modern technology. Radio enthusiasts can still send messages round the world using minuscule wattage and need to show morse proficiency to get a full long wave licence.

HELP A LOCAL ELECTION AGENT ORGANISE THE CANVASS – lists and

more lists, of course, both for pinning on the walls and handing out to the canvassers. Archive, especially with a disc drive, should be able to handle it.

BECOME A MUSIC COMPOSER – the QL is possibly not the best computer on the market for this purpose but Miracle System's Midi music interface is said to be having software redevelopment and could be on the market soon.

CHECK YOUR WINNINGS – write a program for checking betting shop winnings. Programs can be bought of course. Especially useful if you bet by phone and need the information instantly available.

BECOME SELF EMPLOYED – Quill, Abacus and Archive have all the capacity needed to handle your paperwork. For a one-man business *General Ledger* from SD Microsystems makes accounting a doddle. If you are on the dole don't forget the Enterprise Allowance. Freebie help is available too in deciding what business to start.

MAKE A DAY PLANNER WITH AUDIBLE REMINDERS – no reason why this job which is done by electronic pocket organisers should not be done on a static QL for people who work from home. You are on your own with this one however as no commercial program seems to be around.

DESIGN A CALORIE COUNTER – slimmers can find plenty of calorie data to input into a program to count up the day's tally. There is probably a program around to do this but I haven't spotted it.

Shopping

USE YOUR MODEM TO GO SHOPPING – Compustore Online provides a direct cut price shopping service with delivery to your door (ring 0753-868191 for details). Prestel also lists businesses that supply everything from vitamin tablets to home-delivered flowers.

SEND A TELEX ACROSS THE WORLD – again through Prestel. Their Telex Link service allows your computer to output a message that can be delivered anywhere that the teleprinter network reaches. You can also send freebie messages directly to other Prestel users by the mailbox service.

Finally, as an extra suggestion, how would you like to ADD TO THIS LIST. The pooled knowledge of readers of QL World could probably increase it a hundredfold. The Editor, I am sure, would like to see your ideas.

DIY TOOLKIT

I shall now reveal the inner workings of the CLIP functions, introduced last issue, suggesting more uses as I explain the code and its ramifications.

For years I have wanted a 'clipboard' that can read characters from the QL screen, and type them into any other—a kind of OCR for QL Windows, to save me from ever having to read and re-type what the QL already 'knows'. At last I've done it, and the listings show how. After years of conditioning I couldn't resist the tradition of QL puns, and dubbed it *Clipboard*.

Clipboard can scan Qdos or Argos to find and identify all screen and console windows. It reads special attributes like character-size and fonts from the window. With this information the scanner can read or test lines or individual characters, and scoot around any window.

The editor lets you check or change the clipped text; it can be called up at all times, in its own pop-up window. Text is held in string pages of up to 32K; the routines listed share data space between ten pages.

The only important thing about *Clipboard* is to remember to load it. You will keep new finding ways to use it. I mis-typed a line-number while programming it, and lost a long PRINT line from the listing. I still had the latest version in the output window, so I just clipped the characters and fed them straight back into ED.

Squirt

To copy various bits into one *SuperBasic* procedure, LIST the lines containing the bits, clip them up and squirt them back into Basic somewhere else. It's similarly easy to squirt text into *Quill*, *Archive*, editors, emulators and just about any other task that lets you type lines into it.

Clipboard was written in response to years of suggestions from QL users. It is not a replacement for existing facilities, but can be an alternative. Often it fills a gap in the QL armoury, allowing figures, code and text to be transferred accurately between tasks. As well as text, it can easily transfer hex data, *Archive* procedures, *Abacus* formulae, *SuperBasic* or other programs, *Easel* labels and so on...

Clipboard can enervate slothful edi-

tors. Psion and Metacomco block copy operations are painfully slow and limited. It is often easier to clip characters from the screen, go to the new place directly, and spit the text back into the file.

Clipboard does not obscure the screen while scanning, but it only reads one screenful at a time. When you've scanned all you can see without satisfaction you must page the task's display and scan again. Swapping can use a single key-press. The editor can cope with up to 32K, so you can extract 500 lines or more, a screen at a time.



Clipboard can pick up past lines from the command window, but *Super Toolkit's* Alt Enter may remember more, depending on line length. It works if the text is still visible, even when Alt Enter would be confused by single key choices, arrows or deletions, or ESC is pressed accidentally on *Minerva*, denying the line to ALT Enter. I use both, and find Clipboard most useful when past attempts warn me that Alt Enter has reached its limit.

CLIP% and CLIP\$ can recognise user-defined graphic characters as well as text. They are useful accessories for games writers and converters who miss Spectrum functions like SCREEN\$ and ATTR, often used to detect collisions in interactive animated sequences.

The Clip functions are also suitable for quick printouts of text displays. Now you can scan characters from the screen, and translate their codes before printing, if need be. Most printers give best results from a stream of Ascii text. Shaded bit-image graphics tend to be slow and streaky

Simon Goodwin shows how to transfer text between console windows, and demonstrates the CLIP functions for the QL and Thor.

by comparison.

Routines to search fonts quickly and pack any character from the screen into a nine byte font pattern are imbedded in **Listing two**. You could scan graphics into a font, and PRINT two colour patterns of any width, up to nine (or 18) pixels high.

Clipboard recognises four extra key-strokes while other tasks are using the keyboard. The codes are configurable by editing; the names are stored in Ascii text in the compiled task, with easily-recognised labels like '*=EDIT' after each decimal code number. Even after compilation you can change the codes with utilities like *The Editor*, *Spy* or *FEDIT*, by patching new digits into the file. You could use LBYTES, PEEK, POKE and SEXEC, but SEARCH_MEMORY would save time. Just replace the stored codes with your own choices, 0-255.

Codes

I prefer to assign the Clipboard to 'control' codes, 0-31, or non-display codes 192-255, as most tasks ignore them. This discussion assumes the keys set in Listing 1.

Control E calls SHOW_BUFFER, the clipboard Editor, saving the previous screen if RAM permits, before displaying various options and the current clipped text. You'll need to change this code if you need *Spellbound* turned on at the same time, as Control E calls up its examples.

Control P types the next line from the clipboard page into the current task, as if you typed them yourself. The first line is 'next' after the last. The computer burps briefly if it can't find anywhere to put the characters.

Each page can hold up to 32K of text, about 400 lines, but 3K is usually plenty. You can have many pages on one board, and this helps me keep text from different windows apart. This program allocates ten pages, unless you change line 1140 in **Listing one**. To swap between pages, call up the editor, press P then enter a digit, 0-9

Control W highlights each window in turn, so you can select a default. The highlight changes the colours on the screen, but Clipboard can still recognise them all. Change '*=HIGHLIGHT' (Listing one, line 1330) if you don't like my

choice; the range is 0 to 255, and the effect depends entirely on the background colours. 7 and 2 are good alternatives; 17, 34 and 222 are interesting. Press Space to highlight another window, Enter to pick this one, or Esc to escape changes.

Control G turns on a 'clip cursor' in the default window so that you can clip text from it. If the default is not the window you want, press Enter or Esc to turn off the clipper and pick another window with Control W.

You may clip text while a window is highlit, but you are meant to use Control W to pick a default. Visibility is improved and the cursor size is updated if you press Enter to confirm the selection, then Control G to Get text from the window while it is displayed in its usual colours.

Clipboard has its own cursor; you specify the characters that you want to clip by moving the cursor over them. This is controlled by the DIALOGUE routine, which recognises a score of auto-repeating key-strokes.

Cursor

The four arrow keys move the cursor one space at a time, in character-sized steps. You can't move the cursor outside the current window. Alt-Left and Alt-Right move quickly to the start or end of the line.

Shift keys control reading. Shift Right reads the character under the cursor, if possible, and adds it to the current clipboard page, then moves the cursor one place to the right. Control Left and the Thor's grey arrow key move to the left and remove one character from the buffer. Control E displays the contents of the current Qlipboard page.

Shift Enter scans all characters to the end of the line, and puts them on the clipboard, followed by an Enter character. Shift Space is similar but adds a space at the end of the line. In either case the cursor ends up at the start of the next line, if any.

Last month's demonstration routine stripped out duplicate spaces in a line, so that text clipped from a *Quill* screen was not plastered with spaces added by justification or centring. Sometimes margins, tabs and 'extra' spaces are significant, so SCAN_LINE in this month's listing illustrates verbatim scanning, and recognises extra spaces.

The disk Qlipboard has both routines, so you can retain or ignore spaces after the first in each group. If the scanned line consists of nothing but spaces it is condensed to a single Enter character, which gives the same result more economically. Sometimes the default cursor grid does not match the position of text; you may find this if you try to read pixel-positioned labels from diagrams. If text has scrolled or been placed with CURSOR you might have to nudge the cursor to get it right on

```

1000 REMark QL CLIPBOARD by Simon N Goodwin and Phil Spink
1010 REMark Version 6.7; Exts: TURBO TOOLKIT 2, QBASE, CLIP
1020 IMPLICIT w,c,key,windo :DATA_AREA 32 :REMark Needs #1
1030 :
1040 REMark **** QL & Thor keyboard-specific stuff
1050 sv_arbuf=SYSBASE+138 :REMark Key store address
1060 sv_keyq=SYSBASE+76 :REMark Pointer to key queue
1070 REMark Keycodes are ASCII for ease of patching
1080 pickey%="23 *Windows" :REMark Control W
1090 stuffkey%="16 *=Paste line" :REMark Control P
1100 clipkey%="7 *=GET text" :REMark Control G
1110 edkey%="5 *=EDIT" :REMark Control E for Editor
1120 REMark **** Window & page configuration
1130 underline%=0 : getsize=1 : page%=0 : max_windows%=80
1140 maxpage%=9 : REMark MAXPAGE%+1 pages
1150 IF COMPILED : maxline%=2300 : ELSE maxline%=120
1160 :
1170 REMark **** Grab space for data & window details
1180 DIM xshift%(max_windows%),yshift%(max_windows%)
1190 DIM height%(max_windows%),width%(max_windows%)
1200 DIM base(max_windows%),enter$(1),space$(1),newline$(1)
1210 DIM Esc$(1),Line$(maxline%),Page$(maxpage%,maxline%)
1220 enter$=CHR$(10) : space$=CHR$(32) : Esc$=CHR$(27)
1230 newline$="" : REMark Symbolic Newline, suits IO.EDLIN
1240 :
1250 WINDOW 512,256,0,0
1260 WINDOWS :FIND_MYQ :Line$="Qlip 6.7\":"cpos%=1:windo=0
1270 REPEAT indefinitely : PICK_WINDOW : POLL
1280 :
1290 DEFINE PROCEDURE PICK_WINDOW
1300 LOCAL paper%,selected,prior
1310 REMark Result is global WINDO {0 .. global CHX - 1}
1320 WINDOW 512,256,0,0 : selected=0 : REMark Clip Window
1330 paper%="4 *HIGHLIGHT" : REMark XOR 222, 2, 17, 34
1340 WINDOWS
1350 IF windo<win% : first=windo : ELSE first=0 : REMark Guess
1360 REPEAT tilpik
1370 FOR windo=first TO win%-1
1380 SHOW_WINDOW windo,paper%
1390 DIALOGUE windo,selected
1400 SHOW_WINDOW windo,paper%
1410 IF selected<0 AND windo<>first
1420 SHOW_WINDOW first,paper%
1430 SHOW_WINDOW first,paper%
1440 windo=first : REMark Reminisce
1450 END IF
1460 getsize=(selected<>0) : REMark New CSIZE?
1470 IF selected : EXIT tilpik
1480 END FOR windo
1490 first=0
1500 END REPEAT tilpik
1510 END DEFINE PICK_WINDOW
1520 :
1530 REFERENCE happy
1540 DEFINE PROCEDURE DIALOGUE(win,happy)
1550 LOCAL key,endx%,endy%,x%,y%
1560 REPEAT getspec
1570 IF getsize
1580 stepx%=PEEK_W(base(win)+38)
1590 stepy%=PEEK_W(base(win)+40)
1600 END IF
1610 CSIZE (stepx% DIV 4)-1,stepy%>19
1620 POKE_W my_window+38,stepx% : UNDER underline%
1630 POKE_W my_window+40,stepy% : SET_MYQ
1640 POKE_L my_window+46,PEEK_L(base(win)+46)
1650 POKE_L my_window+50,PEEK_L(base(win)+50)
1660 endy%=yshift%(win)+height%(win)-stepy%
1670 endx%=xshift%(win)+width%(win)-stepx%
1680 x%=xshift%(win) : y%=yshift%(win) : REMark Top Left
1690 REPEAT seek
1700 CURSOR x%,y%:CURSOR_ON #1::key=CODE(INKEY$(#1,-1))
1710 SELECT ON key
1720 =192,194 : REMark Move left and maybe delete
1730 IF key=194
1740 IF LEN(Line$):Line$=Line$(1 TO LEN(Line$)-1)
1750 END IF
1760 IF x%-stepx% >= xshift%(win) : x%=x%-stepx%
1770 =200,204 : REMark Move right and maybe clip
1780 IF key=204 : Line$=Line$ & CLIP$(#1)
1790 IF x%+stepx% <= endx% : x%=x%+stepx%
1800 =208 : REMark Move up one character line
1810 IF y%-stepy% >= yshift%(win) : y%=y%-stepy%
1820 =216 : REMark Move down one line

```



```

1830      IF y%+stepy% <= endy% : y%=y%+stepy%
1840      =197 :REMark Pixel shift left
1850      IF x%>xshift%(win) : x%=x%-1
1860      =205 :REMark Pixel shift right
1870      IF x%<endx% : x%=x%+1
1880      =213 :IF y%>yshift%(win) :y%=y%-1 :REMark Bit up
1890      =221 :IF y%<endy% :y%=y%+1 :REMark Down a bit
1900      =193 : x%=xshift%(win) :REMark Zoom to start/end
1910      =201 : x%=endx% - (width%(win) MOD stepx%)
1920      =252,254 : SCAN_LINE
1930      =63 : BEEP 500,0 : temp%=CLIP%(#1) : REMark "?"
1940      REPEAT Intel : IF NOT BEEPING : EXIT Intel
1950      IF temp%<0 : BEEP 1000,-temp%
1960      =232,edkey% : REMark F1/Edkey = Editor & options
1970      CURSOR_OFF : SHOW_BUFFER : NEXT getspec
1980      =10,27,32 : REMark Enter confirms, SPACE = Next
1990      happy=(key=10)-(key=27)
2000      CURSOR_OFF : EXIT getspec
2010      END SELECT
2020      END REPEAT seek
2030      END REPEAT getspec
2040      END DEFINE DIALOGUE
2050 :
2060      DEFINE PROCEDURE SCAN_LINE
2070      REPEAT scan
2080      IF LEN(Line$)<maxline% : Line$=Line$ & CLIP%(#1)
2090      IF x%+stepx% <= endx% : x%=x%+stepx% :ELSE EXIT scan
2100      CURSOR x%,y% : CURSOR_ON #1!
2110      END REPEAT scan
2120      IF key=252:Line$=Line$&space$ :ELSE Line$=Line$&enter$
2130      IF y%+stepy% <= endy% :x%=xshift%(win) :y%=y%+stepy%
2140      END DEFINE SCAN_LINE
2150 :
2160      DEFINE PROCEDURE SHOW_BUFFER
2170      REMark LOCAL mem; DEALLOCATE MEM after BREAK tests
2180      vdu_lines=256 :vdu_start=131072 :REMark Screen 0
2190      mem=ALLOCATION(vdu_lines*128) :REMark Screen buffer
2200      IF mem>0 :MOVE_MEMORY vdu_start TO mem,vdu_lines*128
2210      CLS :UNDER 0 :INK 4 :CSIZE 1,0 :AT 1,0 :PRINT Line$
2220      REPEAT opt
2230      SET_MYQ : OPTIONS editor : IF editor=0 : EXIT opt
2240      XCHANGE Line$,enter$,newline$:CSIZE 1,0:INK 7:AT 1,0
2250      CLS 3:CLS 2 :Line$=EDIT%(#1,Line$,maxline%) :cpos%=1
2260      XCHANGE Line$,newline$,enter$
2270      END REPEAT opt
2280      IF mem>0
2290      MOVE_MEMORY mem TO vdu_start,vdu_lines*128
2300      DEALLOCATE mem
2310      END IF
2320      END DEFINE SHOW_BUFFER
2330 :
2340      REFERENCE txt$
2350      DEFINE PROCEDURE XCHANGE(txt$,old$,new$)
2360      LOCAL p%
2370      REPEAT swap
2380      p%=old$ INSTR txt$ : IF p%=0 : EXIT swap
2390      txt$(p%)=new$
2400      END REPEAT swap
2410      END DEFINE XCHANGE
2420 :
2430      DEFINE PROCEDURE POLL
2440      LOCAL key,keyq,pos%,not_space
2450      REPEAT doze
2460      key=PEEK_W(sv_arbuf)
2470      IF key=pickey% : SET_MYQ : EXIT doze
2480      IF key=edkey% OR key=clipkey%
2490      keyq=PEEK_L(sv_keyq)
2500      REPEAT click
2510      IF key=edkey% : SHOW_BUFFER : EXIT click
2520      DIALOGUE windo,not_space:IF not_space:EXIT click
2530      END REPEAT click : REMark Await ENTER/ESC
2540      IF NOT windo_offset : POKE_L sv_keyq,keyq
2550      END IF
2560      IF key=stuffkey% AND LEN(Line$)>0
2570      POKE_W sv_arbuf,0 : REMark Cancel auto-repeat
2580      nextpos=enter$ INSTR Line$(cpos% TO LEN(Line$))
2590      IF nextpos>0
2600      STUFF Line$(cpo% TO cpo%+nextpos%-1)
2610      cpo%=nextpos%+cpo%
2620      IF cpo%>LEN(Line$) : cpo%=1
2630      ELSE
2640      STUFF Line$(cpo% TO LEN(Line$)) : cpo%=1
2650      END IF : NEXT doze : REMark Dream furiously

```

top of a character, so that CLIP can recognise it. Press Shift and Alt at the same time as any arrow to position the cursor precisely in the window.

The smallest text normally has one blank column on the left of each character, and one blank row at the top. You can test a position by pressing "?"; Qlipboard tries to read the character under the cursor, and beeps once if successful; two beeps mean that the pattern was not recognised in either font for that window; you may need to move the cursor slightly.

If the window uses more than one CSIZE you may need to over-ride the default for Control G, which is the most recent size set. Call up the editor and type X to change the width, or Y to set the height. A cursor appears above the old value. Standard values are 6, 8, 12 and 16, for X, and 10 or 20 for Y. The new setting persists until you change it or select a window with Control W and Enter.

Once you have finished clipping text, press Enter or ESC to turn off the clip cursor. If you forget to turn off the scanner you may confuse it with another task, and have to skip past it when swapping cursors later. Control P will not work while the clip cursor is active, but you can call the Editor directly from the clipper by pressing Control E, or F1 for 'help'.

Taskforce

Users running *Taskforce* or other tools that use standard Qdos windows will find that Qlipboard swaps keyboard queues automatically. Pick the task's cursor and press Enter after loading, so Qlipboard can identify its own queue and get started. POKE SYSBASE+146,9 if you prefer to use TAB instead of Control C.

Qlipboard works very well with *Taskmaster 2.35* despite its quirks, like nine pixel tall names in the 'Files' display. You can clip from all the windows, as long as characters and border lines don't overlap. Note that the 'Info Box' display puts the date on line 6 of the Qlipboard editor, and the 'Notebook' accepts control codes.

Systems with 'extended' console software must rely on their own devices when it comes to queue swapping. Qlipboard needs to be able to move a cursor over the windows of other tasks, but *Qpac-2* prevents this by 'locking' windows so you cannot see through them.

This makes trouble for DIALOGUE (1540), even when 'EXEP""U' ensures an unlocked window. Qlipboard takes careful account of QPac's changes to Qdos channel usage; the editor and 'stuffer' work fine, and the Clip functions happily read a task's own windows with Qpac loaded. It is a pity that current versions separate task displays in such an inflexible way.

The Thor is more accommodating, if you POKE SYSS_VARS +133, -1 before starting, to discourage Argos from grabbing the

display and cursor at odd moments. Press the Sys Req key to swap keyboard queues as necessary; two taps returns to *Xchange* from Qlipboard, with Thor Basic running alongside.

The Thor's PC keyboard gives standard codes whether or not you press Shift with Space or Enter. This eases typing but means Argos users must press Control Enter or Control Space to scan lines; these return the same codes as their shifted QL equivalents. Qdos treats Control Enter like Enter, or Control J, while Control Space corresponds to the Thor's Break key.

Qlipboard clones the fonts in each window, and uses them in the editor, so you see the same characters in the editor as you saw on the task display, as long as the task has not gone, taking the font with it! This feature suits the multi-fonted Thor XVI, and also helps to decode horrid games fonts. If you must know what it really says, clip the lines, pick a standard window, and consult the Qlipboard, to see the text in normal letters and colours.

You could add code to load Psion's subscripts font, or others, when several fonts are used in a window and you can't be sure which is current. MODQUILL, on Quanta's *Psion 1* library disk, should help, but I have been scolded for 'doing it all', so I'll stop there.

The Qlipboard editor uses CSIZE 1,0 text to suit all Qdos fonts. The full window can be reduced for a tv, but Listing one uses the same window for clipping as for editing. The menu needs MODE 4, unless you change the column numbers in OPTIONS or CHOOSE.

TurboQuill Plus uses fixed internal fonts which match the Thor default, rather than the characters indicated by QL channel definitions. The letter f is narrower than on the QL, and the second column of k is lower. This means that CLIP% and CLIP\$ cannot recognise the letters k or f, or some non-Ascii characters, from QL TurboQuill Plus displays. The Thor is fine, as its fonts always match the screen.

Minerva

A keen Minerva user could fix this by extracting the TurboQuill characters with the A1 return from CHAXTOP, which returns the font pattern for any character on screen. Build up and set new default system fonts, if you are determined to Qlip art from TQ+ without nagging doubts!

Type X, Y or U to change the character options after calling up the editor. Set UNDER 1 to read underlined text, or UNDER 0 to distinguish underscores from spaces.

Normally Qlipboard picks up the character-size from the window channel definition; thus it is not limited to the eight standard CSIZES. In fact it copes with single or double height text with any Y step 0 to 49, and X steps from 4 to 19. If windows have

```

2660 END IF
2670 SUSPEND_TASK 15 : REMark Encourage other tasks
2680 END REPEAT doze
2690 END DEFINE POLL
2700 :
2710 DEFINE PROCEDURE STUFF(text$)
2720 LOCAL status%,pos%
2730 pos%=1
2740 REPEAT q_all
2750 status%=QUEUE%(text$(pos% TO))
2760 IF status%=0 : EXIT q_all
2770 IF status%=LEN(text$) : BEEP 500,127 : EXIT q_all
2780 IF status%<0 : status%=-status% : REMark Stuffed
2790 pos%=1+LEN(text$)-status%
2800 END REPEAT q_all
2810 END DEFINE STUFF
2820 :
2830 DEFINE PROCEDURE SHOW_WINDOW(w,c)
2840 LOCAL half%
2850 OVER -1 : IF w<=511
2860 BLOCK width%(w),height%(w),xshift%(w),yshift%(w),c
2870 ELSE
2880 half%=w DIV 2 : REMark Avoid Sinclair's BLOCK 512 bug
2890 BLOCK half%,height%(w),xshift%(w),yshift%(w),c%
2900 BLOCK half%,height%(w),xshift%(w)+half%,yshift%(w),c
2910 END IF : OVER 0
2920 END DEFINE SHOW_WINDOW
2930 :
2940 DEFINE PROCEDURE WINDOWS
2950 LOCAL chbas,chmax,ch,chnp,addr
2960 REMark Fill arrays etc. with 0..WIN%-1 window details
2970 windo_offset=CHANNEL_ID(1) : REMark Check channel
2980 windo_offset=windo_offset-INT(windo_offset/2^16)*2^16
2990 my_window=CHBASE(1) : chbas=PEEK_L(SYBASE+120)
3000 channel0_driver=PEEK_L(PEEK_L(chbas)+4)
3010 windo_offset=chbas+4*windo_offset
3020 windo_offset=my_window-PEEK_L(windo_offset)
3030 chmax=PEEK_W(SYBASE+114) : win%=0
3040 FOR chp = chbas TO (chbas+chmax*4) STEP 4
3050 IF PEEK(chp)<>255
3060 addr = PEEK_L(chp)
3070 IF addr+windo_offset=my_window:NEXT chp:EXIT chp
3080 IF PEEK_L(addr+4)=channel0_driver
3090 addr = addr+windo_offset : REMark Skip extras
3100 base(win%) = addr
3110 width%(win%) = PEEK_W(addr+28)
3120 height%(win%) = PEEK_W(addr+30)
3130 xshift%(win%) = PEEK_W(addr+24)
3140 yshift%(win%) = PEEK_W(addr+26)
3150 IF win%<max_windows%
3160 win%=win%+1
3170 ELSE EXIT chp
3180 END IF : REMark Room for details
3190 END IF : REMark Something like #0
3200 END IF : REMark Open channel
3210 END FOR chp : REMark WINDO_OFFSET, WIN%, MY_WINDOW set
3220 END DEFINE WINDOWS
3230 :
3240 DEFINE FUNCTION CHAN_ID(num%)
3250 RETURN CHANNEL_ID(num%) : REMark TURBO Toolkit
3260 REMark DIY TK: RETURN BPEEK_L(BPEEK_L(48)+num%*40)
3270 END DEFINE CHAN_ID
3280 :
3290 DEFINE PROCEDURE FIND_MYQ
3300 PRINT "Qlipboard:";
3310 REPEAT snoop
3320 CURSOR_ON #1! : Line$=INKEY$(#1,-1)
3330 myq=PEEK_L(sv_keyq) : IF myq : EXIT snoop
3340 END REPEAT snoop : CURSOR_OFF
3350 END DEFINE FIND_MYQ
3360 :
3370 DEFINE PROCEDURE SET_MYQ
3380 CURSOR_ON #1! : IF NOT windo_offset : POKE_L sv_keyq,myq
3390 END DEFINE
3400 :
3410 REFERENCE edflag
3420 DEFINE PROCEDURE OPTIONS(edflag)
3430 LOCAL k$(1)
3440 STRIP 4 : CSIZE 0,0 : INK 0 : AT 0,0
3450 PRINT " Qlipboard 1990 SNG+PRS " : STRIP 0 : INK 4
3460 PRINT " Edit Page=";page%;" Quit RAM=";
3470 PRINT maxline%-LEN(Line$);" " : AT 0,59:PRINT "Under=";

```


DIY TOOLKIT

```

3480 PRINT underline%:" X=";stepx%:" Y=";stepy%:" Zap ";
3490 REPEAT keys
3500 AT 0,83 : CURSOR_ON #1! : k$=INKEY$(#1,-1)
3510 IF k$="q" OR k$=Esc$ OR k$=enter$:edflag=0:EXIT keys
3520 IF k$="e" OR k$=space$ :edflag=1 : EXIT keys
3530 IF k$="u" : CHOOSE 65,underline%,0,1
3540 IF k$="p"
3550   Page$(page%)=Line$ : CHOOSE 39,page%,0,maxpage%
3560   Line$=Page$(page%) : edflag=1 : EXIT keys
3570 END IF
3580 IF k$="x" : CHOOSE 70,stepx%,4,19 : getsize=0
3590 IF k$="y" : CHOOSE 75,stepy%,0,49 : getsize=0
3600 IF k$="z" : Line$="" : edflag=1 : EXIT keys
3610 END REPEAT keys : CURSOR_OFF
3620 END DEFINE OPTIONS
3630 :
3640 REFERENCE thing%
3650 DEFINE PROCEDURE CHOOSE(place%,thing%,lo%,hi%)
3660 LOCAL entry$(2)
3670 REPEAT get_entry
3680   AT 0,place% : INPUT entry$;
3690   IF entry$="" : EXIT get_entry
3700   thing%="0" & entry$ : IF thing%<10 : PRINT " ";
3710   IF thing%>=lo% AND thing%<=hi% : EXIT get_entry
3720 END REPEAT get_entry
3730 END DEFINE CHOOSE

```

moved, remember that a window of 512x256a0x0 can scan any Qdos text, anywhere.

Listing one shows Qlipboard 6.7, about 16K of interpreted or compiled SuperBasic. It uses *Turbo Toolkit* plus DIY Toolkit's QUEUE, BASE and CHAN functions. I have tried to use 'core' Qdos facilities whenever possible, so programmers should be able to get it working with any common mixture of Toolkits.

Line 1020 holds Turbo compiler directives, like the REFERENCE lines before some definitions. The program can be interpreted without them. Line 1150 adjusts the size of the 'page' buffer; the interpreter needs MAXLINE%=120 to keep EDIT\$ or EDLINE\$ inside the minimum size of the SuperBasic buffer. Load or enter a long line to expand the area—or compile the program.

Windows

WINDOWS fills four arrays with the position and size of every window, as well as the base address of each channel's definition (*DIY Toolkit Volume C*). Adjust MAX_WINDOWS% on line 1130 if you need more than 80 windows at once.

DIY Toolkit's CHBASE returns the apparent base of channel details as seen by extension code. WINDOW_OFFSET is zero on a standard Qdos machine, but increases as windowing extensions expand the channel table.

If Qdos seems intact Qlipboard takes over cursor swapping, so you don't have to keep typing Control C. FIND_MYQ and SET_MYQ swap queues by PEEKing and POKEing the system keyboard queue pointer, SV_KEYQ, which steps between the queues at the end of console channel definitions, after each Control C or whatever.

I need to scan characters from window O

and others like it, ignoring file and serial channels. Line 30BO sifts out channels with the same IO.DRIVER vector as O, which is always the first entry in the Qdos table. They use the same TRAP 3 input and output code as SuperBasic's channel 0, so it seems safe to assume that they define windows.

Second

WINDOWS skips its own default channel, 1, so it takes a second Qlipboard task to scan characters from Qlipboard pages. SuperBasic can't run the scanner and the editor simultaneously, but that seldom matters as you can direct lines from the page anywhere. In 1986 I designed *Turbo's* LINK_LOAD to run several modules of a program in parallel; users must compile each 'thread' separately and declare appropriate globals.

CHAN_ID is used to find the index number of 1 in the Qdos table. Unless you've clobbered window 1 this value is always 1 in interpreted Basic, so RET 2^16 will suffice.

Highlight

PICK_WINDOW uses the WIN% array entries to highlight each window in turn. Unless the cursor size is over-ridden DIALOGUE reads it from the channel being clipped, setting the scanning window 1 to use the same CSIZE and fonts.

Line 1930 puts the second processor in the spotlight for a moment. The main chip twiddles its thumbs till Sinclair's 8049, or the Thor XVI's 6802, has played finished the first note.

CURSOR_ON !1 matches Toolkit 2's CURSEN; likewise CURSOR_OFF corresponds to CURDIS. PAN 0,115 and PAN

0,116 do the same jobs for most QL and Thor Basics, a little more opaquely, unless you read *QL World* in February 1989.

Control E calls SHOW_BUFFER, which may reserve buffer memory with ALLOCATION and DEALLOCATE. Toolkit 2 alternatives are ALCHP and RECHP; DIY Toolkit Volume H has RESERVE and DISCARD. Note that ALCHP returns zero (ERR.OK!) if it fails to find the memory requested.

It there's enough free RAM, MOVE_MEMORY saves and restores the default QL screen (32K at 128K). You can do this more slowly with a loop of PEEKs and POKEs, or leave it out if you don't mind overlapping windows.

Editor

As listed SHOW_BUFFER uses Turbo Toolkit's EDIT\$, but you could use DIY Toolkit's EDLINE\$, or EDLIN from the MCS disk rom. The compiled editor lets you select different lines by moving the cursor, but this always winds back to the start, with CPOS%=1. The editor used by INPUT and EDIT is puzzled by lines containing Enter characters, so XCHANGE translates them into backslashes (top right on a QL keyboard). Thus you can split and join scanned lines, even with the rom editor.

STUFF makes determined use of QUEUE% to enter characters; QUEUE% featured in December 1989's *QL World* (DIY Volume Q). You could use Turbo's TYPE_IN instead of QUEUE% to enter characters, but might get stuck or lose text; TYPE_IN is designed for short strings, and does not report errors.

The peppering of calls to CHBASE and SYSBASE should ensure that the code suits all Qdos systems and windowing variations. SYSBASE returns 163840 on most systems, but programs may not run on other machines if they assume that address for the system variables. Thor users can call SYS_VARS instead.

The machine code source in listing two was assembled using HiSoft's *Devpac*. Like all recent DIY Toolkit listings, you can obtain my source, to adapt it or incorporate it into a larger program.

The START routine points A1 at the table of details at the other end of the listing, and calls BP_INIT, the rom vector that adds new functions to SuperBasic. This routine is at the start so it is easy to find after loading the code. It is not used later.

The first big lump is fairly normal parameter-handling, which creates a nine byte buffer on the A7 stack and passes its address, and a channel number, to SD.EXTOP. This points AO at the channel details and calls CHAXTOP, where things get interesting.

First we exclude MEM channels, the only others that allow EXTOP, and check there's room for a cursor at the current window position. Notice how the cursor check

```

* QL WORLD DIY TOOLKIT - QDOS character CLIP functions
* Version 1.1, Copyright 1990 Simon N Goodwin.
*
start      lea.l      define,a1
           move.w     $110,a2      BP.INIT vector
           jmp        (a2)

*
* CLIP% & CLIP$ code - process channel parameter
*
clip_int    moveq     #-1,d7      Flag integer return
           bra.s      clip_param
clip_str     moveq     #0,d7      Flag string result
clip_param  move.w     $118,a2     CA.GTLIN Vector, ensures
           jsr        (a2)        4 bytes of stack space
           bne.s      bad_exit
           subq.w     #1,d3      One parameter expected
           bne.s      bad_param
           move.l     0(a1,a6.l),d0 Get BASIC channel number
           addq.l     #2,a1      Leave room for one INT
           move.l     a1,$58(a6) Set maths stack BV.RIP
           muls       #40,d0     Channel table size
           add.l     $30(a6),d0   Add base offset
           cmp.l     $34(a6),d0
           bge.s      what_chan   Past end of table?
           move.l     0(a6,d0.l),d0
do_extop    bmi.s      what_chan   Negative if closed
           move.l     d0,a0        A0 is channel ID
           lea.l     chaxtop,a2    Address of CH routine
           moveq     #0,d3        ERR.NC if window is busy
           lea.l     -10(a7),a7    Protect some stack space
           movea.l    a7,a1        A1 -> Pattern buffer start
           moveq     #9,d0        SD.EXTOP key
           trap       #3          Call the device driver
           lea.l     10(a7),a7     Discard pattern (this time)
           move.l     $58(a6),a1   Retrieve maths stack BV.RIP

*
good_try     tst.l     d0          Channel error?
           bpl.s      return_val   No
           move.l     d0,d1        Return error code in D1
           moveq     #0,d0        Disregard EXTOP error code
return_val   tst.l     d7          Integer or string result?
           bmi.s      return_int
return_str   tst.w     d1          Did the lookup succeed?
           bmi.s      return_nul   Return null "" if not
           move.b     d1,0(a1,a6.l) Stack character code
           subq.l     #2,a1        Room for string length
           move.l     a1,$58(a6)   Update BV.RIP
           move.w     #1,0(a1,a6.l) Stack length
           bra.s      got_str
return_nul   clr.w     0(a1,a6.l)   Stack a null string
got_str      moveq     #1,d4        Indicate string result
           rts

return_int   move.w     d1,0(a1,a6.l) Put result in space
           moveq     #3,d4        Indicate type is INT
           rts                  Return EXTOP error code
what_chan    moveq     #-6,d0      CHANNEL NOT OPEN error
bad_exit     rts                  Error code is in D0
bad_param    moveq     #-15,d0     BAD PARAMETER error
           rts

range_err    moveq     #-4,d0      Out of Range report code
           rts                  (i.e. no room for cursor)

*
* CHARACTER eXTENDED Operation: A0-> Channel, A1-> Buffer
*
chaxtop      moveq     #40,d0      Size of a MEM block
           cmp.l     (a0),d0      Exclude EXTOP to MEM as
           beq.s      bad_param   it does not suit CHAXTOP
           move.l     34(a0),d0    D0.L := XPOS & YPOS
           bmi.s      range_err    Reject if XPOS < 0
           tst.w     d0            Test Y (in lower word)
           bmi.s      range_err    Reject if YPOS < 0
           add.l     38(a0),d0     D0 := D0 + XINC & YINC
           cmp.w     30(a0),d0     Is the window too short?
           bhi.s      range_err
           swap       d0          Move X to the lower word
           cmp.w     28(a0),d0     Is the window too narrow?
           bhi.s      range_err
find_pixel    move.w     34(a0),d1   D1 := X
           add.w     24(a0),d1     Add window offset to X
           move.w     36(a0),d2   D2 := Y
           add.w     26(a0),d2     Add window offset to Y
           movea.l    50(a0),a2    Get screen RAM base address
           lsl.w     #7,d2        1 line uses 2^7 = 128 bytes
           adda.w     d2,a2        Get address of start of line
pixel_mask    move.w     $8080,d7   Presume MODE 4 pixel mask
           btst      #3,$2(a6)     Check MODE (SV.MCSTA)
           beq.s      mask_ok      MODE 4, no worries
           bclr      #0,d1        MODE 8 or 12, X must be EVEN
           btst      #2,$2(a6)     Test Thor MODE 12 bit
           beq.s      not_a_thor   This must be the QL's MODE 8
           move.w     #$C0C0,d7    Use the Thor 16 colour mask
not_a_thor    move.b     #$C0,d7    Adjust MODE 4 mask for 8 colours
mask_ok       movea.w    #128,a5    Depth of one QDOS video line
           btst      #4,$6(a0)     Double height attribute?
           beq.s      no_taller
           adda.l     a5,a5        Double Y step to suit
no_taller     moveq     #1,d5

```

processes X and Y words together. The QL rom does this, and similar tricks can add four bytes to four others in one step; beware of overflows.

Next I set up registers, ready to scan a grid six pixels wide and nine high, in various steps to reflect the CSIZE. The table shows the register allocation just after the 'strip' background pixel has been read, at SCANLINE. I clipped the table from my notes in *Devpac*.

Efficient machine code uses registers as much as possible. Values in registers can be accessed much faster than from memory, but optimum allocation is a big problem, complicated by processor quirks. This is where human programmers can excel over compilers. The scanner uses all 16 68008 registers; indeed, I run out of data registers and use A4 as a word data value, holding the background colour. This works, as long as you don't want to do much arithmetic with the value.

The code to extract each pixel, around SCAN_ROW, can be faster than in my PIXEL% reading function, as it need not combine the two colour bytes; it is enough to align them with the STRIP data, which is also a pair of bytes. The scanner builds a font pattern in the buffer addressed by (A1).

Pattern

LOOKUP is potentially another EXTOP routine, taking a buffer pointer in A1 and returning a code in D1. It looks for the pattern at (A1) in both fonts, checking all eight bits. As soon as a mis-matched line is found, it steps to the next character. This saves a lot of time, especially if the first line differs from the pattern. DESCENDERS checks the channel's flags and ignores the penultimate line of the pattern, if UNDER 1 is indicated.

I analysed the standard rom fonts in case it would be faster to check lines in some less obvious order, and found that the third and seventh lines in either font most often distinguish characters apart. The difference was small: out of 12,800 comparisons for each row of 161 characters, those rows distinguished 90 per cent, versus 85 per cent for row 0.

The optimisation might help measurably when scanning lots of small letters, but I kept the simpler code. At present pixel scanning takes longer than font searching unless the character is far down font 2, so I think the balance is about right.

The table labelled DEFINE indicates the names and addresses of the commands. I put this last to ease name changes. The assembled code will still work as long as changes are confined to the end of the file.

A configurable compiled Qlipboard task is available, with the programs and text from this month and last, and various bells and whistles besides. Ask for DIY Toolkit Volume 5 - for Screen Reading. Single


```

        btst      #6,66(a0)      Double width attribute?
        beq.s     no_wider
        moveq     #2,d5
no_wider  move.l    d1,a3
        moveq     #7,d2
        and.w     d1,d2
        lsr.w     #3,d1
        add.w     d1,d1
        move.w    0(a2,d1.w),d4
        lsl.w     d2,d4
        and.w     d7,d4
        move.l    d4,a4
        Save background for later

*
* Scan all nine lines and compare them with the STRIP colour in A4
*
        moveq     #8,d2          DBRA through 9 lines
scanline  add.l    a5,a2          Advance to next line
        move.w    a3,d1          Reset X to left edge
        moveq     #0,d6          Clear fount pattern
        moveq     #5,d3          DBRA count for X (5-0)

*
* Scan a row of six pixels. A2 -> video RAM line, D1 is pixel X, 0-511
*
scan_row  moveq     #7,d4          Mask for bottom bits of X
        and.w     d1,d4          X && 8 is pixel # in word
        move.w    d1,d0          Copy old X before message
        lsr.w     #3,d0          Offset on line is X DIV 8
        add.w     d0,d0          Address only even words 0-126
        move.w    0(a2,d0.w),d0  D0.W := relevant screen word
        lsl.w     d4,d0          Align pixel in Most Sig bits
        and.w     d7,d0          Mask out the background
        cmp.w     a4,d0          Does pixel match the strip?
        beq.s     roll_on        If yes, leave bit unset
        addq.w    #2,d6          Set one fount bit
roll_on   add.w     d6,d6          Shift pattern along
        add.w     d5,d1          Advance to next column
        dbra     d3,scan_row      Stuff fount line in buffer
        move.b    d6,(a1)+
        dbra     d2,scanline      Retrace to start of buffer
        lea.l     -9(a1),a1

*
* Look up the character pattern at (A1) in both founts
*
lookup    moveq     #0,d1          Clear high bytes of code
        move.b    (a1)+,d4        Pre-read the first line
        move.l    42(a0),a2       A2 -> Fount 1 base
        bsr.s     scan_fount
        beq.s     result_d1       Match?
        move.l    46(a0),a2       A2 -> Fount 2 base
        bsr.s     scan_fount
        beq.s     result_d1
        moveq     #-7,d1          NOT FOUND error code
result_d1 moveq     #0,d0          Signal no QDOS error
        rts

*
* SCAN_FOUNT; corrupts D0, D1, D2, A2, A3, A4; Exit: NZ, D1=ASCII
* Entry: A2 -> Fount base, A1 -> Pattern+1, D4 = start of pattern
*
scan_fount move.b    (a2)+,d1      Get first pattern # in fount
        moveq     #0,d0          DBRA pattern count in bytes
        move.b    (a2)+,d0        D0 := Number of patterns - 1
test_start cmp.b     (a2)+,d4      Exclude B6-BB% of matches
        bne.s     zoom_on        Ignore the rest if mismatched
        moveq     #5,d2          DBRA count for more lines
        move.l    a1,a3          Scan pattern at (A3), saving A1
        move.l    a2,a4          Check versus (A4) preserving A2

scan_all  cmpm.b    (a4)+,(a3)+    Check each line of pattern
        dbne     d2,scan_all      First 7 lines are Ok...
        beq.s     descenders     Skip remainder of pattern
zoom_on   addq.l    #8,a2          Assume next ASCII code
        addq.b    #1,d1
        dbra     d0,test_start
        moveq     #-7,d0          Not found, exit with 7 reset
        rts

*
descenders btst     #0,66(a0)     Check for UNDER 1
        bne.s     last_line      UNDER 0, Check line 8
        move.b    (a4),d2
        cmp.b     (a3),d2
        bne.s     zoom_on
last_line move.b    1(a4),d2      Line 9 (last in pattern)
        cmp.b     1(a3),d2
        bne.s     zoom_on
        rts

*
* Routine names and addresses
*
define    dc.w     0,0            No procedures
        dc.w     2              Two functions
        dc.w     clip_int-*
        dc.b     S,'CLIP%'
        dc.w     clip_str-*
        dc.b     S,'CLIP%'
        dc.w     0              End of list

*
end

```

Sinclair QL World DIY TOOLKIT February 1991, Listing 2 - CLIP code

volumes cost £7, recorded on disk or microdrive. Each further volume ordered at the same time costs £3 extra; there are now 14 volumes. CGH supplies disks, but tape users must send one formatted cartridge for each volume required. Call Richard Alexander on (0559) 384574 or write to *DIY Toolkit*, Cwm Gwen Hall, Pencader, Dyfed, Cymru SA39 9HA.

Fast

The disk Qlipboard incorporates Phil Spink's fast windowing editor, with adjustable tabs, word-wrap, paging, filing and other civilised features. That editor was enhanced and tested for *Quickfax* more than a year ago. That project has stalled but Phil's editor deserves an airing. It has been carefully and intuitively designed, and recognises a superset of popular QL control keys.

Character scanner register utilisation table

- *D0 - Scratch
- *D1 - Current pixel X co-ordinate
- *D2 - line count, 8-0 top to bottom
- *D3 - column count, 5-0 left to right
- *D4 - Pixel bit offset in video word, 0-7 left to right
- *D5 - X step between pixels: 1=CSIZE 0/1, ?; 2=CSIZE 2-3,?
- *D6 - Accumulated character form for this line
- *D7 - Pixel mask for this MODE (WORD)
- *A0 - Pointer to channel definition block
- *A1 - Pointer to byte buffer of 9 lines in QL fount format
- *A2 - Pointer to start of current video RAM line (Y2)
- *A3 - Leftmost pixel X co-ordinate
- *A4 - Background pixel colour, normalised
- *A5 - Y step between lines: 128=CSIZE ?, 0; 256=CSIZE ?,1
- *A6 - Pointer to QDOS or Argos system variables
- *A7 - SUPERVISOR stack pointer



SOFTWARE FILE

INFORMATION

Program: Anelpum Quat
Price: £8.00 disk or mdv (& 80p for p&p etc.). All QLs.
Publisher: CGH Services,
 Cwm Gwen Hall, Pencader,
 Dyfed, Cymru SA39 9HA.

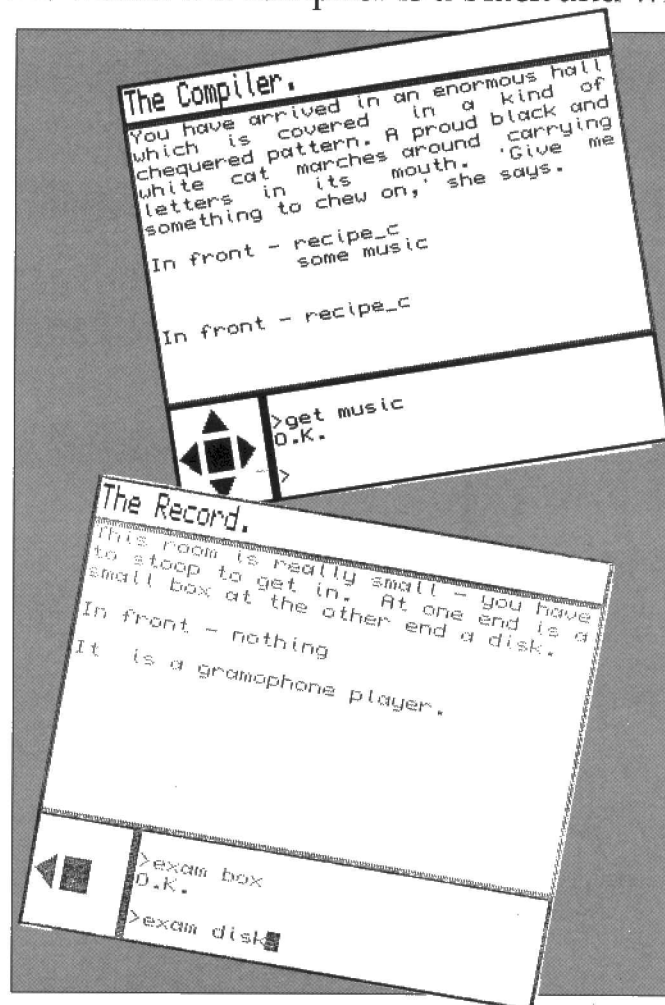
ANELPUM QUAT

Rich Mellor gets inside the last word in adventure, where the compiler is a black and white cat

This game is a new adventure from the CGH stable. This represents yet another new direction in adventures, being set inside the very computer you are using. The manual (as usual) is short and gives just a little insight into the game, before you enter your first location.

As soon as you load the game, some of the differences in this adventure will become apparent. The screen is divided into four, with the name of the current location at the top of the screen, below which appears a description of the location. At the bottom of the screen is a graphical representation of the current exits, and another window where you enter your commands. The method of entry of commands is distinctive in that the parser will only allow you to type words which it recognises. It does this by checking every letter as it is typed, and if the given letter does not fall into a recognised word, a beep is issued and the letter is not shown on screen. This may speed up checking of the commands, and also prevents any 'not understood' errors. However, the method of deleting letters is rather cumbersome in that instead of the previous letter being deleted, a back arrow is printed on screen. This can lead to confusion, but thankfully pressing ESC abandons the command.

One soon becomes used to the new method of entry of commands, and with the program's ability to understand multiple commands, one soon begins to zip around the locations. Movement can be done by the normal east, west, north, sound commands, or simply



by using the cursor keys if preferred. I must admit that I would have preferred a normal means of command entry, especially when you begin and are unsure which commands the computer will understand.

This first location actually presents the first major problem. You are outside the computer which is not working, and need to get inside it to fix it. Without a screwdriver or manual to hand, you soon discover that there is no easy method of entry; more 'direct' methods must be used. Once

inside the computer, you will find some text telling you that the computer is in a right state with lots of unfinished tasks and half-written routines. As you go around, you will discover just what these tasks are.

Unfortunately at first appearances there do not seem to be very many locations, and even fewer objects. Looking around you will find lots of different characters which make up the different parts of the computer, ranging from compilers to viruses. Each of these has a different function,

and while some will be able to help you, others are there to hinder you (or even kill you).

The compiler (a black and white cat) is a great help during the early part of the game, although presenting him with something to compile makes him ask you a cryptic question which must be answered to get anything useful for him. Answering some of these questions will open new areas of the computer up for you to search, while others will provide you with the means to finish tasks. A sign near the entry point to the computer tells how well you are doing by presenting a reading of a failure rate. I have only achieved an 82% failure rate so far in several hours of play and so I have a long way to go yet!

Many of the different tasks involve plays on words to give you a hint as to what is needed (there is no help feature in this adventure). For instance, the stack is missing its pointer - do not try looking for a computer-type pointer!

The puzzles within the game are very well thought out and involve a logical means of solving them. It provides a little insight into the workings of a computer as well as being an entertaining game. Fortunately there is no need to have an in-depth knowledge of computers, although a *Roget's Thesaurus* can be a big help. Thankfully the game can be saved onto any device to return to at a later date, since I can see that a pen and paper may be just as useful as a computer keyboard. Despite one or two minor criticisms about the method of command entry, I must say that the program is well worth getting one's teeth into.

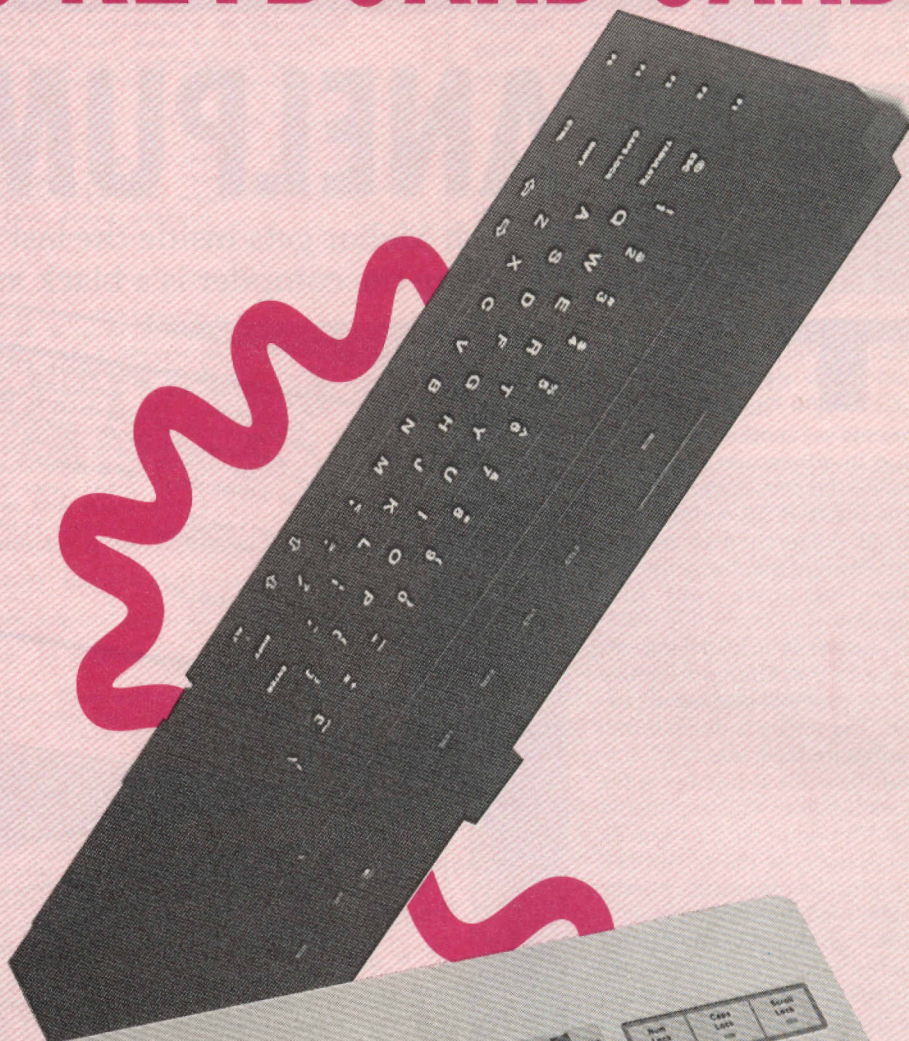
PRODUCT REVIEW

PC KEYBOARD CARD

Many QL users prefer to use a PC-style keyboard. Mike Lloyd tests a new keyboard interface card.

Some people in the computer market must be psychic. Just as my trusty Sandy keyboard was beginning to show signs of its venerable age the postman brings a saviour to the rescue – a new piece of electronic wizardry from Jeorgen Falkenberg. Falkenberg, you will remember, was responsible for the excellent scanner reviewed in *QL World* some months ago. His latest offering is not a keyboard, but a chip-laden link to a standard PC-style keyboard, which gives users a choice of the full range of PC keyboards available in the UK today at very reasonable prices.

The original QL keyboard came in for a lot of criticism early in its life. One journalist wrote "Hands up all those who like the new QL keyboard . . . what a lot of funny-looking hands". Designed to be cheap to make and better to use than the Spectrum's rubber keys (and, incidentally, much better than the awful keyboard of the IBM PC jr.), the QL keyboard was still not as good as a "grown-up" PC keyboard. I



touch-typed my way through the manuscript of my first book using nothing but a QL keyboard, and I know of other authors who did the same, but many good typists could not get used to its tacky feel.

Smart money

One solution was to replace the top of the QL with one possessing better-sprung keys, and this proved to be a much more acceptable improvement than might have been thought. Nevertheless, the smart money went on add-on keyboards which were linked to the QL via a cable and which had a separate numeric keypad and additional keys laid out in PC XT style. As the owner of one of the original Sandy keyboards I can vouch for its ease of use and considerable endurance. Giving a journalist a keyboard is rather like giving a six-inch nail to a sixteen-stone carpenter – it tends to get thoroughly hammered.

Recently, as I remarked earlier, the Sandy keyboard began to show signs of senility. It would forget some of the keypresses, and occasionally had a stroke which would paralyse one side completely. The ribbon cable which lay across my desk like the Channel Tunnel rail link across Kent would snag papers and books so that a slight movement of the keyboard would partially dislodge it, rendering many of the keys useless until the socket had been pushed back into place. Keyboard bounce – the QL's annoying habit of printing random letters twice – was not eradicated by the Sandy keyboard, although it appeared to be reduced. All of these weaknesses could be lived with, but the keys in the centre of the board were beginning to wear out so that a delicate press was ignored and only a table-shaking stab would produce a response. After four years of unrelenting toil, it was time to find a new keyboard.

Letterbox

With impeccable timing, Jeorgen Falkenberg's gizmo was thrust through my letterbox. Mounted on a card some 12cm by 6cm, a cluster of eleven silicon chips gave some clue as to the computing power required to accept input from a PC keyboard and translate it so that it was fully compatible with what the QL expects. At one end of the keyboard a short cable was mounted, ending in a DIN plug familiar to anyone who has examined the back of a PC, or a QL for that matter. At the other end of the small card was an empty mounting slot. Unfortunately, the Jiffy bag contained nothing else – no fitting instructions, no user guide, and no invitation for a week's expenses-paid holiday in the Rhine valley.

The purpose of the card was obvious enough – but what about fitting it? The QL's keyboard is managed by an interesting little chip called the Intel 8049. It was

perhaps the first "computer on a chip", in that it has its own arithmetic logic unit (alu), central processing unit (cpu), a read-only memory (rom), and a tiny amount of access memory (ram) all built on the same wafer of silicon. For a chip so comprehensively equipped, it is a little disappointing to find that it is not very bright. In the QL its task is to relieve the M68008 cpu from much of the mundane work of listening out for keypresses and controlling microdrive and diskdrive activity. The empty mounting slot on the keyboard controlling card was exactly the size for an Intel 8049, and small cutouts in the card fitted exactly with the topology of that part of the QL's motherboard where the Intel 8049 is situated – towards the front left, immediately beside the microdrive housings.

With only a little trepidation I opened the QL's case, carefully avoiding dislodging the wires which supply current to the various leds on the computer's top. The ribbon cable to the Sandy keyboard was quickly removed and, using a plastic chip-removing tool and a lot of patience, so was the Intel 8049. I pressed the chip into its new home on the piggy-back card and then began the process of gently persuading the pins underneath the card to slot into the holes left by the Intel chip. The secret is to slowly increase pressure across the entire width of the mounting so that pins do not become bent. It is surprising how much weight is actually required to press home the pins.

The final problem was how to get the cable out of the QL case so that a keyboard could be plugged into it. After seriously considering the removal of the second serial socket for a few moments I finally decided to unplug the expansion card, thread the cable through the huge hole in the QL's left side and then slot the expansion card back in place.

Some days later I received a call from Falkenberg's UK agents to say that the QL-Keyboard-90 card was also available in a form which simply plugs into the QL's rom port. This is a much simpler method of upgrading the QL and one which is recommended to anyone frightened of taking a screwdriver to his or her beloved computer.

Simpler method

The rest was easy: simply rob a passing PC of its keyboard and pug its DIN plug into the waiting socket. Nevertheless, powering up the QL was a little nerve-racking. What if this wasn't a keyboard controller after all, but some other arcane device which I had completely misidentified? All was well, though. The keyboard lights flickered, the QL booted normally and I began typing away.

It was quite unusual to find myself using a QL with a fully-functioning PC keyboard. Deleting was performed not by the Ctrl-

left arrow key combination but by the PC's backspace key. Similarly, the PC's delight key performed the action normally accomplished by pressing Ctrl-right arrow. The function keys, numeric keypad and cursor keys worked exactly as would be expected. Rather surprisingly, so did the Escape key: it acts like Ctrl-Space to halt a SuperBasic program or exit a line-editing sequence. Within a few moments the feel of the new Keyboard lost its alien nature and I quickly adjusted to the new key positions.

Free to choose

QL-Keyboard-90 is for sale in the UK at £75, which includes VAT, and it is available from TK Computerware in Kent. At the price it is not cheap, and many users without their own PCs will have to consider buying a keyboard as well, so can it compete in the keyboard add-on stakes?

The answer must be a definite yes. The keyboard mapping vastly exceeds in power and ease of use anything previously offered on the QL market. By buying Falkenberg's card you are then free to choose any of the myriad of PC keyboards available through mail order in Computer Shopper and other similar magazines. Good PC keyboards start at around £50, although some de luxe models can cost nearer £100. For many users, though, it will be a case of borrowing a keyboard from a PC, or maybe buying a second-hand keyboard attached to a broken PC, reducing the costs. Two simple jumpers on the card allow the firmware to be configured for old or new styles of XT and AT keyboards.

With full documentation and in its final release version, QL-Keyboard-90 has the feel of a quality product and it will undoubtedly improve your typing. Running Digital Precision's excellent *PC-Conqueror* emulator couldn't be easier if you are using a real PC keyboard. People with a desk cluttered by too many keyboards can now make one keyboard do the work of two. All in all, QL-Keyboard-90 is a complex product which does a very good job without bothering the user and if it suits your circumstances, and your pocket, I recommend that you give it serious thought.

INFORMATION

Hardware:	QL-Keyboard-90 PC-compatible add-on keyboard controller
Price	£75 inc VAT
Manufacturer	Jeorgen Falkenberg West Germany
Available from	TK Computerware, North Stanford, Ashford, Kent. Tel: 0303 81 2801

Programming

C

In the last instalment we considered symbolic constants, variables, sequence control and conditional statements. There is another conditional statement (the switch statement) that we will look at in a future article.

The two main topics this month are handling arrays and basic file processing. There are three program listings included to illustrate the new concepts. A few other new ideas are also mentioned in passing.

Listing one illustrates very simple array processing. It takes input from the keyboard and counts the number of times that each of the different keys is pressed. When you press the ESC key it displays the results.

The program begins by defining a few symbolic constants. `SIZE_ASCII` represents the highest character code value used in the QL character set — all character codes are in the range zero through 255. `NUM_LINES` represents the number of lines of data that can conveniently be displayed in the window in which Digital C programs run.

Digital C supports int and char data types. You can also declare arrays of integers and characters. The syntax is:

```
data_type variable_name
[array_size];
```

The array_size must be enclosed within square brackets. The number of elements in the array is one more than the array_size. In our program the array contains 256 elements (`SIZE_ASCII + 1`).

Arrays in C are similar to arrays in SuperBasic, but there are a few differences. As they say, variety is the spice of life.

Each element of the array is referenced by specifying a subscript enclosed in square brackets (SuperBasic uses round brackets). For example, the first element in the array is referenced as `variable_name [0]`, the second as `variable_name [1]` etc.

The program in listing one

declares an int array named `num_char`. This array consists of 256 elements numbered from [0] through to [255] (`SIZE_ASCII` is 255). Each element of the array can hold an int value. The array is used to represent the 256 possible ASCII characters.

After printing a few lines of introductory text the program executes a for loop. At the beginning of the loop the value of `sub` is set to zero. The loop keeps repeating so long as `sub` is less than or equal to the highest subscript value in the `num_char` array (255). At the end of each pass through the loop the value of `sub` is incremented by one.

The body of the loop consists of a single statement which sets the value of `num_char[sub]` to zero. If the elements in the array were not initialised to zero they would contain garbage values.

Andy Wright continues his beginners' guide to C with arrays and file processing.

An experienced C coder might have coded the for loop that initialises the array as:

```
for ( sub = 0;
      sub <= SIZE_ASCII;
      num_char[sub++] = 0 )
;
```

In this new version the body of the for loop includes no statements — the single semicolon at the end indicates this. At the end of each repetition of this non-existent body of statements, the following is executed:

```
num_char[sub++] = 0
```

The position of the ++ symbol is important. It tells the system to set `num_char[sub]` to zero and then increment the value of `sub` (ie it uses the current value of `sub` and then increments it).

`Num_char[++sub] = 0` would have incremented `sub` first and then set `num_char[sub]` to zero.

The new version of the for loop is more concise and, in the

eyes of some, more elegant. Such shorthand techniques are common in C programs. People who are new to C sometimes find it confusing. People who are experienced C programmers occasionally take satisfaction in how simple it is when you get the hang of it but how confusing it all can be to the uninitiated.

The next part of the program uses a while loop to process the next key typed in at the keyboard. The ASCII code of the key is stored in the variable `inp_char`.

The loop stops when the (ESC) key is pressed. Each pass round the body of the while loop adds one on (using the ++ operator) to the `num_char` array element whose element number is the same as the value of `inp_char`. So, if you press A the value of `num_char[65]` is incremented by one — the ASCII code for an upper case A is 65. The statement:

```
++num_char[inp_char];
```

illustrates that char variables can be used as integers since they actually contain integer ASCII character codes.

The use of the `putchar` function within the loop makes sure that the character that you type is also displayed on the screen.

The last part of the program

Array declared as:

```
char inp_val[9];
```

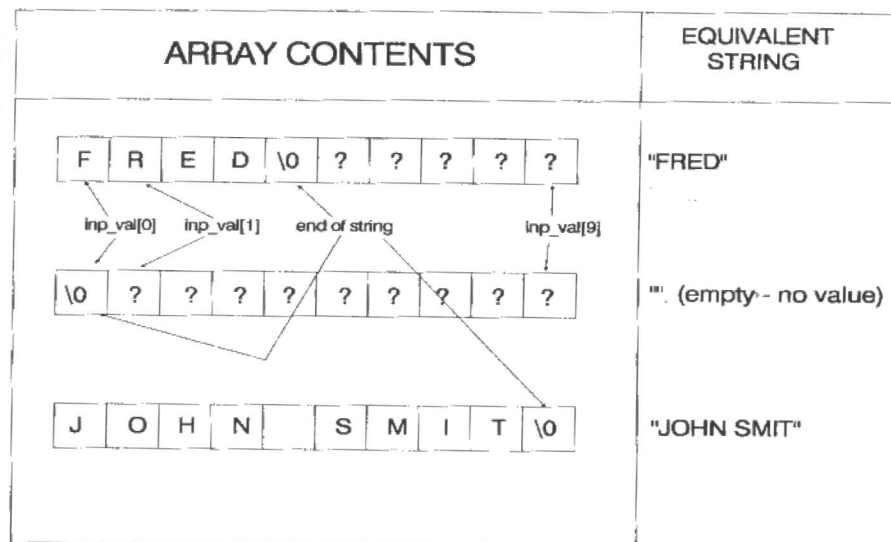


Fig 1: Character arrays and strings

uses a for loop to print out the values held in the array. Initially the variables sub and count are set to zero. The variable sub represents each element number in the array — corresponding to the Ascii characters 0 through 255. The count variable is used to keep track of how many lines have been printed on the screen. The loop keeps going while sub <= SIZE_ASCII — ie until all of the elements in the array have been processed. At the end of each pass round the loop, both sub and count are incremented.

The printf function prints two versions of sub. One is a decimal version of the Ascii character code (using the %3d format), the other is the QL character which has the corre-

sponding Ascii code (using the %c format). On each pass round the loop sub represents and Ascii character. The value of num_char[sub] represents the number of times that the corresponding keyboard character was typed. This is printed as a decimal (%5d) value.

Whenever the value of count is greater than NUM_LINES you will be prompted to press a key. This stops the display scrolling off the screen before you can read it. If you press the ESC key, a break statement is executed. This breaks (jumps) out of the loop that you are in — even though the condition defined in the for statement is still true. Program control passes to the statement at the end of the loop (the printf function in our program). Similar facilities

are provided for getting out of loops in SuperBasic. If you press anything other than ESC the value of count is reset to zero and the for loop carries on with the processing.

Right at the end of the program, getchar is used to make sure that the display remains on the screen until a key is pressed. Note that the value returned by getchar is thrown away or discarded since no variable name has been specified to hold the value.

Exercise One

1. Change the program so that the array elements are not initialised to zero and run it a few times.

2. What would happen if you tried to update num_char[256]

— or any other element beyond number 255? If you try this it will have unpredictable and potentially nasty results since it will update bits of the computer's memory which lie somewhere beyond the end of the array. Accessing non-existent elements in an array is a sure-fire (and fairly straightforward) way of guaranteeing that your program produces garbage — or worse.

3. Try using the alternative for loop for initialising the array elements. Code it correctly and then try coding it incorrectly (putting the ++ operator in the wrong position).

4. Re-write the loop that displays the results so that it doesn't print anything at all for array elements with a value of zero — ie those representing

```
#define SIZE_ASCII 255
#define ESC 27
#define NUM_LINES 10
main ( )
{
    int  num_char[SIZE_ASCII], /* An array of integers */
        sub,
        count;

    char inp_char;

    printf ("This counts the number of occurrences of each ASCII\n");
    printf ("character that you type, until you press (ESC)\n\n");
    for ( sub = 0;
        sub <= SIZE_ASCII;
        ++sub )
        num_char[sub] = 0;

    while ( (inp_char = getchar() ) != ESC )
    {
        ++num_char[inp_char];
        putchar(inp_char);
    }
    for ( sub = count = 0;
        sub <= SIZE_ASCII;
        ++sub, ++count )
    {
        printf ("%3d  <%c>  %5d\n", sub, sub, num_char[sub]);
        if ( count > NUM_LINES )
        {
            printf ("\nPress a key to continue (ESC) to stop\n\n\n");
            inp_char = getchar();
            if (inp_char == ESC )
                break;
            count = 0;
        }
    }
    printf ("\nPress a key to finish");
    getchar();
}
```

/*

Listing 1

*/

PROGRAMMING IN C

keys that were pressed zero times.

5. You should find that the display is messy for the character whose Ascii code is 10 (the ENTER or newline key). This is because printing the character (%c) version forces the screen display to do a linefeed. Try to stop this from happening. You will need to include an if statement to change what happens when sub is equal to 10.

6. What would happen if the same key was pressed more than 32767? Change the code so that this cannot happen.

7. Try printing the value of num_char without specifying an element number.

Listing two illustrates the use of a character array in C. The use of the #include compiler directive and a few new standard library functions are also considered.

The program itself inputs two numbers and calculates the result of multiplying them together. The results are displayed on the screen.

The program in listing two starts off with a #include compiler directive. The basic syntax is:

```
#include file_name
```

The C pre-compiler reads the contents of the file named in the #include directive and includes it in your program in place of the #include directive. The file f1p1_stdio.h that is used in our program is supplied with the Digital C compiler. Most of it is made up of some standard defines. Some of the coding in listing one depends on the #defines of stdio.h being included in the program.

Using #include can save a lot of repetitive coding. Just code the stuff in a single file and use #include to copy it into any program that you like. Just about all C compilers are provided with an stdio.h file although on non-QL computers it is usually called stdio.h and to include it you would use:

```
#include <stdio.h>
/* including stdio.h on a non-QL system */
```

Note that with Digital C the file_name must include the device name (fep1_ is used in

our program; if you are using microdrives then use mdv1_ or mdv2_). On many other implementations of C the device name (or its equivalent) is not needed if you enclose the name between < and > symbols.

The program defines two arrays of characters — each with seven elements — numbered from zero to INP_SIZE.

In C all strings are made up of arrays of characters. This is also true of SuperBasic. However, it is often possible to ignore this fact when writing programs in Basic — you can never really afford to forget it when using C.

One area where C character arrays (or strings) are significantly different from SuperBasic is that in C the end of a string is always represented by a character with the Ascii code zero. The Ascii character with the code zero is represented, in programs, as \0. This means that a character array of 10 elements (numbered from zero to 9) could contain a string of 0 to 9 characters and still have room for the \0 string terminator.

Even when you type in a string of characters enclosed in quotation marks, C treats it as an array of characters and puts in a \0 terminator for you. Figure one illustrates this.

Meanwhile, back at listing two, note that the condition used in the while loop is always true since the expression (1) is always non-zero. This means that the loop will keep on going until it is stopped by a break statement.

The for loop is nested within the while loop. It reads characters from the keyboard and stores them in the character array (string) inp_string_1. The integer str_pos is used to reference each of the elements of the inp_string_1 array in turn. At the beginning of the for loop str_pos is set to 0 — to reference the first element. The condition associated with the loop uses getchar to store the keyboard key into the variable inp_char. If inp_char is the Ascii code for the (ENTER) key then the loop is terminated. The \n used in the program is equivalent to the Ascii code produced by the QL (ENTER) key.

If inp_char is a non-digit then it is ignored. Otherwise, the digit is stored in the string array and the value of str_pos is

incremented. If the array has only one unused element left (ie the value of str_pos is equal to the array_size) the break statement is used — this terminates the for loop immediately. If this check on the value of str_pos was not included then you would be able to type in a potentially unlimited number of numeric digits. As str_pos went beyond the end of the memory allocated to the array, the contents of other areas in the machines memory would get corrupted. Notice that if str_pos equals INP_SIZE the final array element has not been used — the last element is needed so that we can store the C string terminator (\0) at the end of the array.

At the end of the for loop the value of str_pos is checked. If it is zero then no digits were typed in. The break statement then terminates the while loop and the program ends.

If some characters were typed into the array then an extra character \0 is added to the end of the array. In this case str_pos points to the array element after the last one that was entered. All C standard functions recognise the end of a string of characters by the existence of this terminator.

Because of the way that we have built up the array of characters (using getchar) you should find that you cannot change any of the characters that you type in. As it is written the for loop will simply ignore you if you use the CTRL key or any of the arrow keys. But at least all of the alphabetic keys etc are ignored as well.

The value of inp_string_2 is picked up using the fgets C library function. Another for loop could have been used, but the program has gone for versatility rather than consistency. (In most real-life programs it is usually safer to stick with consistency!) The syntax of fgets is:

```
fgets ( character_array_name,
        number_of_characters,
        file_descriptor );
```

In fact the character_array_name is really a pointer to a character string. We will cover pointers in detail in a future article.

The characters that are typed in are automatically stored in the named array. Input is auto-

matically terminated either when the you press ENTER or (number_of_characters - 1) keys have been pressed. You should find that the way that fgets works on the QL allows you to make changes to the string that you type using the arrow and CTRL keys. Whatever you type is also automatically displayed on the computer screen without you having to use putchar.

The fgets function can read from any open device or file. This means that you must tell it where to read data from. The device is specified using a thing called a file_descriptor — sometimes known as a file pointer. In listing two the file_descriptor stdin is used. A file_descriptor is a bit like a channel number in SuperBasic. In C stdin refers to the standard input device. By default this is the keyboard. The stdin device is automatically opened ready for use when a compiled C program is run.

We have already used one function that always reads from stdin (the getchar function). Because getchar can only ever read from stdin you don't tell it which file to read from. With fgets you have to specify a file_descriptor, otherwise it doesn't know where to read its data from. In our program the descriptor stdin is used in the fgets function. When stdin is mentioned explicitly in a program, its value must be defined. The #included file stdio.h includes a definition for stdin so we don't need to worry about it.

One curious aspect of fgets is that if you press ENTER to terminate the input then the \n (newline) character is included at the end of the array of characters that you type. If (number_of_characters-1) are typed in and fgets stops all by itself, then no \n is added to them.

The standard string terminator \0 is automatically added to the array — just after the \n or the last character that you typed. This means that if you type 123 (ENTER) the array contains:

```
123\n\0
```

If you type 12345 (ie INP_SIZE - 1 characters) the array contains:

12345\0

There is nothing in the fgets function to stop you typing non-numeric characters. These will just be included inside the array.

The printf statement at the end of listing two prints the input strings. Both inp_string_1 and inp_string_2 are printed using the %s format specification. %s is used to print character arrays (strings). It starts at the beginning of the array and stops when it gets to the string terminator \0.

The %6d format specification is used to print the result of the expression:

```
atoi(inp_string_1) * atoi(inp_string_2)
```

The atoi function converts a \0 terminated string of Ascii characters into an integer. The conversion stops when any non-numeric character is found within the string (for example a \0 string terminator or a \n newline character). This would mean that if you typed 12a4 the integer calculated by atoi would be 12. If you typed a3 then atoi would return a value of zero!

Note that the * included between the quotes in the printf statement is printed as an asterisk. The * in the list of expressions that are to be printed performs the multiplication of the integer version of each of the input strings. The result of this is printed using a %6d format specification.

After the results have been

printed, the end of the while loop is reached and you should be prompted to enter another number.

Exercise two

1. Change the program so that no \0 is added to the end of the character array at the end of the for loop. See what difference this makes to the way that the program works.

2. Check to see if fgets and atoi cope with negative numbers.

3. What happens if you type in numbers that are too large? Can you change the code easily to stop this from happening?

4. You should find that the display of inp_string_2 produced by the printf function is not always consistent. It de-

pends on whether or not the fgets function added a \n character to it. Change the code so that any \n is replaced by a \0 before the value of inp_string_2 is printed.

5. Try to re-write the statements:

```
inp_string[str_pos]=inp_char;
++str_pos;
```

as a single statement which achieves the same overall effect.

6. Try to replace the bits of the printf statement that prints the value of inp_string_1 by a for loop that uses the putchar function to print the contents of the inp_string_1 array — it should stop as soon as it gets to the \0 character.

What would happen if you

```
#define INP_SIZE 6
#include flpl_stdio_h
main ( )
{
    char inp_string_1[INP_SIZE],
          inp_string_2[INP_SIZE],
          /* Two arrays (strings) of characters */
          inp_char;
    int str_pos;

    while ( 1 )
    /* This condition is always true */
    {
        printf("\nJust Press (ENTER) to quit");
        printf("\nEnter 1st number : ");
        for ( str_pos = 0;
              (inp_char=getchar()) != '\n' ;
              /* No actions on each repetition */
            )
        {
            if ( isdigit(inp_char))
            {
                inp_string_1[str_pos]=inp_char;
                ++str_pos;
                putchar(inp_char);
                if (str_pos == INP_SIZE )
                    break;
            }
        }
        if (str_pos == 0) /* No characters entered */
            break;
        inp_string_1[str_pos] = '\0'; /* String terminator */
        putchar('\n');
        printf("\nEnter 2nd number : ");
        fgets(inp_string_2, INP_SIZE , stdin);
        printf( "\n\n%s * %s = %6d\n\n",
                inp_string_1,
                inp_string_2,
                atoi(inp_string_1) * atoi(inp_string_2) );
    }
}
```

/*

Listing 2

*/

PROGRAMMING IN C

forgot to make it stop at the end of the string (if you try it out you will have to reset your QL before carrying on — unless you can kill off the program somehow — you have been warned.

7. What happens if you forget to include the stdin file descriptor in the fgetc function?

Listing three illustrates how you can process data from files other than the standard input and output devices.

The program given in listing three prompts you to supply a file name and then tries to display the contents of the named file on the screen.

The first few lines of the program are made up of #include and #define compiler directives.

MAX_FILE_NAME is defined to be big enough to hold the largest sized file name available on the QL (in fact it is slightly larger than necessary). LINES_PER_PAGE is the number of lines from the input file that are displayed at a time. CHARS_PER_LINE is the number of characters that fit on a line inside the window that Digital C programs usually run in.

The other symbolic constant that is defined is FILE — it is given the value int. Its use is described below.

As we have already seen, files in C are accessed by using file descriptors. These file descriptors are a bit like SuperBasic channel numbers. In fact, file descriptors are used to access files and devices such as the keyboard or windows on the screen. So are SuperBasic channels.

Before a file or device can be accessed in C, it must be opened and linked to a file descriptor. Like other implementations of C, Digital C provides some devices that are automatically opened and linked to file descriptors. The file descriptors for the devices that are opened automatically are defined for you inside stdio.h.

They are:

stdin
stdout
stderr

By convention, stdin is linked to the keyboard whilst stdout and stderr are linked to the screen. Stderr is usually used to display error messages.

Digital C is rather curious in that all three of the default file descriptors are linked to a single QDOS channel. This channel is a console. You can read from a console (it picks up data from the keyboard) and write to a console. (the output appears in a window on the screen). This means that you can get away with writing programs that read from stdout and write to stdin. Even though you can get away with doing this — you should try to overcome the temptation. Attempting to read from an output device just seems wrong!

A few of the standard C library functions that we have used have written to the default devices. For example, getchar reads a character from stdin, while printf writes to stdout.

In listing two we saw that the fgetc function can read from any file. The cost of this extra versatility in fgetc is that you have to tell it which file or device to read from. We forced it to read from stdin.

The program in listing three defines its own file descriptor and uses it to read data from a file on a disc or microdrive.

The first line of the main function declares the file descriptor. It uses:

```
FILE * file_d;
```

The word FILE is a symbolic constant — defined as int, so the declaration of the file descriptor could have been written as:

```
int * file_d;
```

so why wasn't it?

Most implementations of C define a FILE data type. Digital C doesn't. In most C implementations there would be no need to define FILE as a symbolic constant because it is already defined for you.

In Digital C you could edit your own stdio.h file to include the definition of FILE as int — then you wouldn't have to define it within the program. Just include stdio.h. This would make the whole program look more like a C program as coded on, say, a computer running Unix or MS-DOS. In Digital C, file descriptors should be of data type int, but this isn't necessarily true of other systems. Using the data type FILE gets round this prob-

lem fairly neatly because it can be defined differently in stdio.h on different machines and it doesn't affect your program code.

If the last few paragraphs bother you, it doesn't much matter if they don't make a great deal of sense if you are only interested in using C on a QL with the Digital C compiler. If you are keen on writing C programs that can easily be compiled and run correctly on other computers using different C compilers then it is important to see the significance of what has been said!

The other outstanding point it — why is there an asterisk (*) in front of the file descriptor name file_d? One answer to this question is that there has to be an asterisk in front of it! File descriptors are, in reality, pointers to things. This means that the file descriptor points to something that tells the system where a file is. File descriptors are also known as file pointers. When any kind of pointer is declared in a C program its name must be preceded by an asterisk. We will look at pointers in a lot more detail in a future article. For the time being just include the asterisk.

After the file descriptor declaration, a few integers and a character array are declared. The character array is used to hold the name of the file that is to be processed by the program. The array is declared to be at least large enough to hold the longest file name that you would find on a QL.

A while loop is used to get the file name. Just before the while loop is entered, the file_d descriptor is set to NULL. The symbolic constant NULL is defined in stdio.h (in Digital C it has the value of -32768, on other systems it may be different but it doesn't matter!)

So long as the value of file_d is equal to NULL the while loop keeps repeating and you keep getting asked to enter a file name. We shall see what is so special about NULL in a few moments.

Note that the message that tells you to enter a file name is produced using the fprintf function. This is like printf except it can write to any open file or device. In listing three the message is sent to the stdout device. By default this is the computer screen.

The fgetc function is used to read the filename from stdin. Immediately after the fgetc function there is a check to see if any characters were entered. Just to be different, the strcmp function is used. Its basic format is:

```
strcmp (
character_array_name or
characters in quotes,
```

```
character_array_name or
characters in quotes );
```

The parameters (or arguments) to strcmp are really pointers to strings. Character array names and strings of characters enclosed inside quotes are actually pointers to strings — but more on pointers later.

The strcmp function compares the strings and/or character arrays. If they are equal to each other it returns a value of zero. See the Digital C documentation for other possibilities.

If the statement uses strcmp to compare the inp_name character array with the string \n. Remember that if you just press ENTER without typing any characters then fgetc will insert the characters \n into the character array. If the result of the strcmp is zero then the exit function is used. The exit function terminates the program and closes any open files.

The strcmp part of the program could have been coded as:

```
/* assuming inp_val has been
declared as an int */
int_val = strcmp ( inp_name,
"\n");
if ( int_val == 0 )
    exit ( );
```

The method used in listing three is more compact since it doesn't need to make use of the variable int_val.

Yet another way would have been to adapt the approach used in listing two:

```
if ( inp_name [0] == '\n' )
    exit ( );
```

Note that the == operator cannot be used to compare strings of characters — it only compares ints and char in Digital C. This means that the \n must be enclosed in apostrophes — if it were enclosed in

ordinary quotation marks it would be a string! Inside apostrophes it means the Ascii code of the newline character.

If some characters were typed into `inp_name` the next part of the program uses a for loop to replace any `\n` character in the string by a `\0`. As we have already noted, `fgets` likes to put a `\n` at the end of the string that it gets. The for loop in the program processes each character in the array until it reaches the end of the string (marked by the `\0` character). Any `\n` that is found within the string is replaced by a `\0`. This effectively gets rid of the `\n` and marks a new end to the string. For example:

```
f1p1_fred\n\0
```

becomes

```
f1p1_fred\0\0
```

So far as C is concerned, the new string simply contains the characters `f1p1_fred` since the first `\0` marks the end of the string.

Once this has been done an attempt is made to open the file. This is done using the `fopen` function. The basic syntax is:

```
file_descriptor_name = fopen (
file_name,
```

```
open_mode );
```

The file descriptor must be declared appropriately at the beginning of the program. The `file_name` and `open_mode` can each be either the name of a character array or a string of characters enclosed in double quotes. (If you guessed that

they should be pointers to strings — you guessed right).

The `open_mode` can be one of the following:

`r` - opens the file to read data from it

`w` - opens the file to write data to it, if the file already exists then its contents are deleted

`a` - opens the file to add data to the end of it, if the file doesn't already exist then a new empty file is created

The mode could be the name of an array containing the characters `r\0` or `w\0` etc or it could be "`r`" or "`w`" etc (enclosed in quotes).

If the `fopen` statement fails then the `file_descriptor` is set to `NULL`. If `fopen` succeeds it sets the `file_descriptor` to a non-`NULL` value.

The `fopen` function will fail, for example, if you specify an invalid device name at the beginning of the file name (eg `md1_` instead of `mdv1_`) or if there is no disk in the drive or if you try to open a file for reading and the file doesn't exist.

Our program opens the file for reading. If `fopen` fails and the `file_descriptor` is set to `NULL` then a message is displayed. At the end of the while loop, the descriptor `file_d` is still `NULL` so it goes round the loop again.

Once the file has been successfully opened and `file_d` is not `NULL` the while loop terminates.

The file's contents are then processed within a for loop. The variables `num_lines` and `num_chars` are set to zero — since nothing has been displayed yet. The loop is repeated

```
#include f1p1_stdio_h
#define MAX_FILE_NAME 50
#define FILE int
#define LINES_PER_PAGE 16
#define CHARS_PER_LINE 73
#define ESC 27

main ()
{
    FILE *file_d;
    int inp_val,
        num_lines,
        num_chars,
        sub;
    char inp_name[MAX_FILE_NAME];
    file_d = NULL;
    while ( file_d == NULL )
    {
        fprintf(stdout, "Enter filename ((ENTER) to quit): ");
        fgets ( inp_name, MAX_FILE_NAME, stdin );
        if ( strcmp(inp_name, "\n") == 0 )
            /* No inp_name entered */
            exit();
        for ( sub = 0;
              inp_name[sub] != '\0';
              ++sub)
            if ( inp_name[sub] == '\n' )
                inp_name[sub] = '\0';
        /* Replace newline by end of string marker */

        file_d = fopen ( inp_name, "r");
        /* Open the file for reading */
        if ( file_d == NULL )
            fprintf(stdout, "\n\nUnable to open file\n\n      <??>\n\n",
                    inp_name);
    }

    /*
                                Listing 3 part 1
    */
```


PROGRAMMING IN C

so long as the following expression is true:

```
(inp_val=getc(file_d)) != EOF;
```

The `getc` function is used to read a character from the file associated with the descriptor `file_d` — this is the file that has just been opened. The `getc` function is rather like `getchar` except you must use a file descriptor to tell it where to read data from. The value that is retrieved is stored inside the variable named `inp_val`. If you check the program listing you should find that `inp_val` is declared as an `int` — not as a `char`. This is not a mistake — it

is absolutely vital!

The `for` loop carries on repeating so long as `inp_val` is not `EOF`. What is `EOF`? Perhaps you've worked out that it must be a symbolic constant that is defined for you in `stdio.h`. It is! In Digital C `EOF` is given the value `-1`. (Some other implementations of C use different values, but this doesn't matter.)

When you use `getc` to read data from a file and there is no data left (because you have reached the end of the file) Digital C arranges it so that a value equal to `EOF` is passed back to your program. The value of `EOF` has to be different from all possible real char-

acter values — so it has to be outside the range 0 through 255. If `EOF` was the same as a valid ASCII character code it would be very difficult to work out if you'd reached the end of the file or just a funny character part way through the file. This is why `inp_val` must be declared as an integer. If it was defined as a `char` you would get problems. `Char` variables can only hold values in the range 0 through 255 — they cannot hold `-1` (`EOF`). This means that if `EOF` was passed back to a `char` variable it would end up being converted to a value somewhere in the range 0 to 255. Because of this it would

always not equal `EOF` — and the `for` loop would never end.

It is easy for a newcomer to C to declare characters that are being read from files as `char` instead of `int`, but if you do then you can expect problems! You won't be able to spot when the end of the file is reached.

Inside the `for` loop the `putc` function is used to copy the character that was read from the file to the `stdout` device. The rest of the `for` loop is concerned with stopping the display after a screen full of data has been displayed — rather than letting it all scroll off the top of the screen. The basic logic used is as follows:

```
putc('\n', stdout);
for ( num_lines = 0, num_chars = 0;
      (inp_val = getc(file_d)) != EOF;

    )
{
    putc(inp_val, stdout);
    if ( inp_val == '\n' )
    {
        ++num_lines;
        num_chars = 0;
    }
    else
    {
        ++num_chars;
    }
    if ( num_chars == CHARS_PER_LINE )
    {
        ++num_lines;
        num_chars = 0;
    }
    if ( num_lines == LINES_PER_PAGE )
    {
        fprintf(stdout, "\nPress a key to continue - ESC to quit\n");
        if ( fgetc(stdin) == ESC )
            exit();
        num_lines = 0;
        num_chars = 0;
    }
}
fclose (file_d);
fprintf(stdout, "\nPress a key to quit");
inp_val = getchar();
}
```

/*

Listing 3 part 2

*/

1. If the character is a newline (`\n`) add 1 to `num_lines` and set `num_chars` to zero - since no characters have been displayed on the new line.

2. If `num_chars` equals `CHARS_PER_LINE` the line is full so add 1 to `num_lines` and set `num_chars` to zero.

3. If `num_lines` equals `LINES_PER_PAGE` then the screen is full. Output a suitable message and wait for a key to be pressed. If ESC is pressed then exit the program, otherwise set `num_lines` and `num_chars` to zero and carry on around the loop to display another screenful.

At the end of the loop the file is closed using `fclose`. The syntax is:

```
fclose ( file_descriptor );
```

Once the file has been closed the file descriptor could be used to open another file.

Exercise three

1. What would happen if the `\n` was not removed from the file name that was input using

`fgets`? Try it out.

2. What happens if you miss out the quotes from the "r" in the `fopen` function?

3. What happens if you try to get characters from a file that has not been successfully opened?

4. the `fclose` function actually returns an integer value. A value of zero means that the file was successfully closed, -1 means that it wasn't. Amend the coding so that it checks that the file was closed properly. One way of checking that this works is to write a program that tries to close a file that has not been opened. Picking up the value returned by `fclose` involves using:

```
integer_variable = fclose (
file_descriptor );
```

5. In listing three the final for loop reads characters and displays characters. Try to re-write the whole thing so that it reads lines and displays lines using `fgets` to read them and `fprintf` to write them out again. You will need to define a character array in which to read the lines - and fix it so that the call to `fgets`

never reads more than, say, 73 characters (a screen line full). You need to know that the `fgets` function returns a value of NULL if the end of file was reached.

```
integer_variable = fgets ( .... );
if ( integer_variable == NULL )
{ /* END OF FILE REACHED */
}
```

6. Change the program so that it counts the number of words and lines in the input file.

7. Write a program that does the following:

a) Gets an input file name from `stdin` and opens the file for reading.

b) Gets an output file name from `stdin` and opens the file for writing.

c) Copies the whole of the input file into the output file.

Be careful when you test this program - remember that if the output file already exists its contents will be deleted. It is probably best to play with a blank disk or microdrive cartridge.

Change the program so that it appends the contents of the input file to the output file.

C is a much more versatile language than we have so far seen. Some of this extra versatility will be covered in future articles.

For example it is possible to get a C program to read `stdin` data from a file instead of from the keyboard - and to write `stdout` data to a file rather than to the screen. All of this can be done without having to re-write the program at all. This process is sometimes called redirection (reading `stdout` from somewhere different from usual and writing `stdout` to somewhere different). One of the few areas in which the MS-DOS operating system is more convenient than Qdos is in the way it allows the redirection of `stdin` and `stdout`. In a future article we'll look at how this can be achieved using Digital C.

It is also possible to take the `stdout` output from one program and pass it into the `stdin` of another program. This process is sometimes known as piping (the output of the first program is piped into the input of the second). Again, we'll take a look at this in more detail later.

The Advanced Wordprocessor for the Sinclair QL text⁸⁷

text⁸⁷ version 3.00 and **text⁸⁷budget**, offer today's state-of-the-art user-friendly environment for document production. Extremely fast in operation with documents of any size and capable of mixing and displaying different type styles and sizes both on the screen and on paper. Supports multiple columns, multiple rulers, and multi-line headers and footers. With integrated spelling-checker in three languages, editable dictionaries and extensive support for highest quality printing possible on daisywheel and 9-pin printers. Budget version cannot be used with 24-pin, laser and deskjet printer drivers. Upgrade to the full version will be available at the price difference.

text87: £60 • text87budget: £45 • founttext88 + fountedit89: £40

2488: £15 • typeset90-deskjet: £20 •

typeset90-laserjet: £40 • typeset90-GQ5000: £40

Send for our new comprehensive leaflet

Software available in English, French, German and Italian. Prices are inclusive of airmail. Payable by cheque, Postal Order or Eurocheque. Please specify language and disk system (all 3 1/2" and 5 1/4" formats can be supplied). text87 requires at least 256K memory expansion.

Printer Drivers

text⁸⁷ comes with about 30 sophisticated printer drivers for dot-matrix and daisywheel printers supporting all the international characters available on individual printers and proportional spacing, double-width, condensed, italics etc. if available. In addition, you can add any of the following drivers for state-of-the-art output not available from any other QL program.

founttext⁸⁸ and **fountedit⁸⁹**, the graphic printer-driver and editor for **text87** provide high quality graphic founts without the limitations in text editing and document size imposed by QL desktop-publishing programs. More than 40 quality founts of up to 72 pixels in height are supplied and new founts can be designed or captured from saved screen images. For Epson and compatible 9-pin and 24-pin printers.

24⁸⁸, text-mode printer drivers for selected Epson LQ, NEC P, Star LC and XB and Panasonic KXP 24-pin printers offer advanced features such as multiple typefaces, proportional spacing, micro-justification, double height, shadow and outline modes (depending on the printer model).

typeset⁹⁰ is a set of three separate text-mode printer-driver products: **typeset90-deskjet** for Hewlett-Packard Deskjet, Deskjet Plus and Deskjet 500 support printer's internal founts plus the full range of Roman, Helvetica and other plug-in cartridges. **typeset90-laserjet** and **typeset90-GQ5000** are for Hewlett Packard laserjet and Epson GQ-5000 printers supporting a very large selection of founts in different sizes.

Software⁸⁷, 33 Savernake Road, London NW3 2JU

SOFTWARE FILE FILE EXECUTIVE

Bryan Davies tries a thoughtfully-constructed new file-handling utility with a reasonable price-tag.

INFORMATION:

Program: File Executive
V1.00

Price: £12

Supplier: J. O'Connor

16 Parsons Way
Wootton Bassett
Wiltshire SN4 8DA

The instructions are quite comprehensive, and the available functions are what one would now expect, after having had the use of Toolkits for several years; some of the functions are a bit unusual, though. The command format is much similar to that used in

memory.

The similarity of style to The Editor extends to the start-up and Help screens. Help is clearly laid out, and sufficiently descriptive without occupying too much screen space. You can manage without the written instructions. The help file is not loaded into memory, so the program disk needs to be in the appropriate drive for help to be available.

The commands available in-

```
r [device] {[specifier]}
```

where "r" indicates read, "[device]" is the device name in the usual form — `flp1` or `flp1_will` do — and "[specifier]" is for a string within a file or files. If a specifier is present, the directory displayed will be of only those files which contain the string. The instructions are a bit misleading here, in that the specifier string has to be a part of the file name, rather than

```

  1. ARCHPRM      14. LIB5_PRO      27. STATPRF_DDD
  2. BOOT         15. LIBGUIDE_DEF    28. STATPRF_EXE
  3. BOOT_PRO     16. LIBGUIDE_DOC    29. STATPRF_EXE
  4. CLONE        17. LIB_GCH    30. STATPRF_EXE
  5. DEF_clone_bas 18. LINPRF_EXE      31. STATPRF_EXE
  6. DRAFLINE_bas  19. LINSOLVE_DDD    32. TRANSFER_PRO
  7. DRAFLINE_demo 20. LINSOLVE_EXE     33. ---DRAFLINE---
  8. DRAFLINE_DOC  21. LIST_PRO        34. ---LINSOLVE---
  9. GUIDE_SFPRF   22. PRINTER_DAT     35. ---STATPRF---
 10. LIB1_PRO      23. STATPRF_EXE
 11. LIB2_PRO      24. STATPRF_EXE
 12. LIB3_PRO      25. STATPRF_EXE
 13. LIB4_PRO      26. STATPRF_EXE

```

[illegible]

File Executive is a file-handling utility, and comes on 5¹/₄in or 3¹/₂in disk only. Instructions for use (11 pages) are supplied in both a *Quill* and an *Editor* document on the disk with the program. The program file is compiled *Super Basic* and requires a *Turbo Toolkit* extensions file to be loaded before it. A run-time set of extensions is loaded automatically by the boot, unless you choose to by-pass it because you already have an equivalent file loaded. The provided commands work with microdrives or floppy drives, but not with hard disk.

The Editor — mainly mnemonic, single-letter, with parameters where appropriate. The interface with the user is through three windows, the main one to display requested information, and smaller, single-line ones above and below it to give basic information on the medium in use and to allow the input of commands to give error messages.

The program file is about 36KB in size, and takes a total of around 50KB when in use. This size limits it to being the only job on a basic 128KB QL, but it shouldn't restrict normal operations on a system with plenty of expansion

clude the usual Directory, Copy, Delete and Format, but not Rename. In addition, you can View the contents of files, Find and Exchange strings within files, Get information on a file, Set the Dataspace allocated to a program file, View the Status of a medium, and Print the Current Directory. Command files (as in Editor) can be used, commands can be chained one-after-the-other, and they can be repeated.

The single-letter commands are easy to remember, being in mnemonic form. For example, the format of the command for displaying a directory on the screen is:

being part of the file itself as implied by the description of the function. It is a pity the square brackets are used in the opposite sense to that many users will be familiar with; that is, the presence of `[]` indicates a mandatory parameter, rather than the more-usual optional one. The curly brackets are used to enclose optional items; using square brackets within curly ones seems unnecessary. Several commands can be entered in sequence, separated by semi-colons. For example:

r / flp2;bz

will cause a directory of all

files on flp2 to be displayed, followed by a buzz from the QL's magnificent speaker. Commands — including several on one line — can be repeated by preceding them with the figure for the number of repeats, as in <10(bz;/flp1)> to make the speaker buzz ten times, with the directory of flp2 being displayed after each buzz. An indefinite repeat is obtained by using <rp> — <rp bz> will keep the speaker buzzing continuously. All this will be familiar to the users of The Editor.

The QL function keys are used for familiar operations; F1 brings up help information, F2 recalls the last command used, F4 rewrites the screen, and F5 enables the program main window size to be altered and the position to be changed, using the cursor keys, alone and with ALT. Commands can be aborted during execution by pressing ESC.

Parameters can extend the utility of commands considerably, and a decent selection is offered. Numbers, text or

You can specify a range of files (by number) to be copied; when only one file is specified, you can also give the file copy a different name from the original. The same format is used with the Delete command. Files can be viewed; they normally pass through the window in a continuous stream, but the up-cursor key pauses the flow, and ESC stops the operation. Finding and Exchanging Strings within files is a similar operation to Viewing files, but the specified string will be displayed in inverse video whenever it is encountered, and the number of occurrences of the string will be displayed when the operation has been completed. The format for a find and exchange is:

```
es [filenumber] [device]
[source] [dest]
```

where "[source]" and "[dest]" are the string to be found and the one to replace it by, respectively.

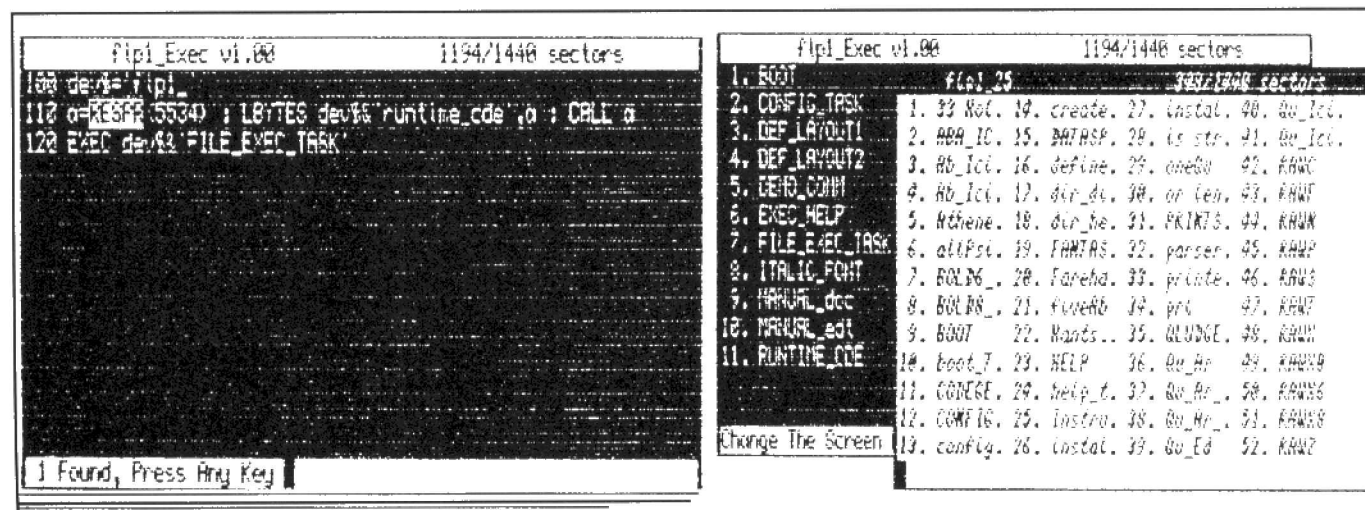
The Getting and Setting Information function reads the

the medium label, write status (protected or not), total and available storage space in sectors and kilobytes, number of files present, number of files selected (by a previous command), search string (specified with the Read/Directory command).

Command files can be written with any WP or editor program, provided you can ensure the resulting file does not contain unwanted control codes. Quill is suggested as being usable for this purpose; presumably you would have to Print the file to disk/cartridge with no printer-driver present. The supplied demonstration command file is shown in the illustration, which also shows the alternative display found and another window position. As can be seen, several lines of commands are possible. Comments can be inserted into a command file by prefacing the line with an asterisk. You can also display a prompt such as "Insert disk into flp1" in the command window, and cause the command file to be

played, with truncated file names being terminated with a full stop to show that they are incomplete. The display colours can be altered within the basic QL range of 256 "colours"; this is, perhaps, only a frill, but continuous use of a utility can lead to irritation with certain colours or combinations. Somewhat unexpected in a cheap utility is the facility to change the case of file names, the options being lower-, mixed- or upper case. The Sort facility allows files to be sorted into alphabetical order when a directory is read. Subsequent copying will be done in the displayed order, not the actual one on the medium.

A routine CONFIG_TASK is provided, to enable the user to specify the name of the defaults file, which is read when the program starts running, and the location of the help file. Error messages from the program, and from QDOS, are displayed in the small, lower window. The instructions explain the program messages (11 of them) and you are referred



strings can be used, strings having to be enclosed in quotation marks. Quotation marks are also needed when characters are used which the program would otherwise recognize as "for other purposes"; for example, a name which includes spaces must be enclosed by quotes. Files can be identified by Ranges, such as 1,3,5 or 10-20, or a mixture of the two forms. Parameters can be separated by full stops, spaces, commas or slashes.

The Copy command includes the Wild Card option, and the standard Y/N/A/Q (yes/no/all/quit) query, including when overwriting is necessary.

file header and displays the file name, length, date of creation, type (job or data), and dataspace (if the file is a job). Combined with this command is one to permit the user to alter the allocated dataspace for a job. The Format command is basic, and accepts no parameters other than the device name; you are asked to confirm that you wish to proceed with the format.

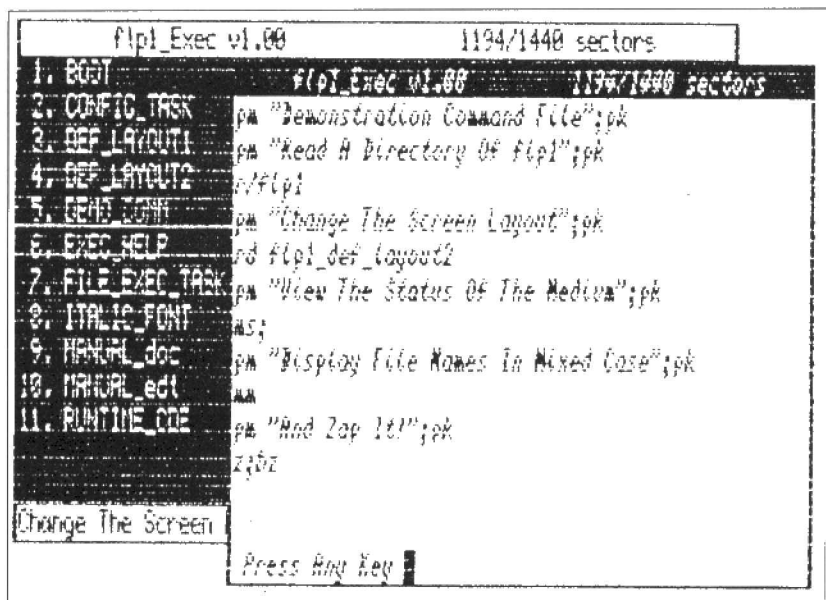
It is a pity no provision was made for giving the medium a Volume Label, nor for formatting to anything other than a default number of sectors. Viewing the Status of a medium produces a display of

paused until the key is pressed.

There is a Zap command, which clears information on the current medium from memory (without doing anything to the medium!) The currently displayed directory can be sent to a printer (the port can be specified) and printed in the same format as it is displayed, using an 80-character width. The buzzer command (bz) could be handy if the user wants warning of when a long command sequence has been completed. The number of columns in the window display defaults to three, which is probably the best choice for most users, but the illustration shows four dis-

to The QL User Guide, Concepts section, for explanation of QDOS messages. The program did not "fall over" when errors occurred during the review.

File Executive worked alongside several other EXEC-ed jobs on the review system without apparent interaction. It is not a pop-up job, as the comparable program Files (from Sector Software) is, but you can use CTRL-C to return to it from the current application, rewriting the display using F4 if necessary. The displaying of a basic directory is done quickly, the default display being three columns wide by 13 rows deep



columns, ink and paper colours, Copy speed (in terms of the basic QL methods of copying and the use or otherwise of the Overwrite query), sort facility on/off, and the device name for the default fount file.

One operational problem was noted. When the program was quit, in the prescribed fashion (using the <qu> command), in order

to restart it with the default settings, it refused to load; initially, the message from QDOS was "out of memory" (the free space reading was 175Kb) when the command <EXFILE_ EXECT_ ASK> was used, changing to the old-familiar Turbo one of "Task...halted, after line 0..." when the command was expanded to <EXECFLP1_ FILE_ EXECT_ ASK>. Yes, I know I

should remember the answer to that one! The average user will not be aware of what to do next, though. Maybe, some day, someone will properly sort out the problems associated with extensions which are "present", but not "correct". Another minor problem was that the demonstration file moved the window and changed the screen fount, and you would have to use both the Fount and F5 commands to get back to the settings (assuming you remembered the position and size figures).

All-in-all, the program is a workmanlike job, operates smoothly and speedily, and has little to its debit. Thoughtful touches, like ensuring that the results of a command are displayed immediately, without the user needing to take further action, show that the writer has given serious thought to the project. The price is reasonable; no doubt there are still some users who do not have any of the well-known utility programs, and they should give this one serious consideration when deciding to make a purchase.

— that is 39 files at a time. Resizing the window to the maximum screen size increases the potential number of files displayed to 96. No doubt many users will be irritated by the way the QL's built in DIR command, and the add-on commands of other utilities, restrict the number of files displayed simultaneously, and File Executive does a better job here. Some long file names will

get truncated when the default window size is used, and if you use the option to increase the number of display columns above the default of three, but the user has the freedom to set whatever number of columns and the window size that best suit the file names used. A defaults file allows the user to save these settings — screen X and Y co-ordinates and dimensions, number of displayed

columns, ink and paper colours, Copy speed (in terms of the basic QL methods of copying and the use or otherwise of the Overwrite query), sort facility on/off, and the device name for the default fount file. One operational problem was noted. When the program was quit, in the prescribed fashion (using the <qu> command), in order

QL SUPERTOOL KIT II by Tony Tebby

Over 118 Commands:- Full Screen Editor, Key Define Print Using, Last Line Recall, Altkey, Job Control, File Handling, Default Directories, Extended Network, 16k Eprom Cartridge Version@£24.15d Configurable Version on Microdrive@£24.15d

MIRACLE SYSTEMS PRODUCTS

DK Disc Interface (Inc Tool Kit II)@£99.82b QL Centronics Printer Interface@£28.75d QL Expanderam 512K Thru-card@£99.99e

QL HARDWARE

Single 3.5" Disc Drive & (Own PSU)@£103.50a Dual 3.5" Disc Drive & (Own PSU)@£188.60a 3.5" DS/DD Discs - 10 off@£9.20c Q POWER REG. The only real solution to you QL overheating (switched mode power supply run cold)£24.15c QL Keyboard Membrane@£11.50d QEP III Advanced Eprom Programmer@£121.90d Care Eprom Cartridges each@£5.75c ULA Chip ZX6301@£15.64c

PRINTERS

STAR LC 10 Mono@£179.40a STAR LC 10 Colour@£224.25a CITIZEN SWIFT 24-pin Colour@£356.50a

SOFTWARE 87 (State MDV or Disc)

TEXT87V.300 (Requires Min 256k)@£60.00d FOUNTEXT 89@£15.00c FOUNTEXT 88@£25.00c TEXT 87/FOUNTEXT 89/FOUNTEXT 88@£94.99b 248B PRINTER DRIVER@£15.00c Upgrade to Text 87 V.2.00. Return old copy together with@£25.00d

MONITORS (Price including lead)

Philips BM7522 Amber Hi-Res@£97.75a Philips CM6833 Colour Med-Res@£253.00a Philips AV7300 TV/tuner for above@£69.00b Philips BM7502 Green Hi-Res@£97.76a

HOW TO ORDER:

ALL PRICES INCLUDE VAT

By Post. Enclose your Cheque/PO made payable to CARE Electronics or use ACCESS/VISA. Allow 7 days for delivery

TONY TEBBY SOFTWARE (QJUMP)

QLFP (Micro/P disk interface upgrade)@£14.95d QMON II@£23.00d QTYPE Type/Spell Checker@£29.09d QPAC 1-Desktop accessories, calander, alarm, calculator, typewriter, digital clock sysmon@£21.85d

ZITASOFT SOFTWARE by Steve Jones

LOCKSMITHE copies M/DRIVE - M/DRIVE@£14.95c 4MATTER + LOCKSMITHE copies M/DRIVE - DISC@£23.00c

The above programs are not for use in the UK
SIDEWINDER - High resolution printer driver prints full screens or parts of screens from postage stamp size to large banners. Prints sideways, invert, scale, mirror, text insertion@£19.95c
SIDEWINDER PLUS - (for expanded QL's) includes all the features of above, PLUS multiple label printing, desktop publishing files and printer driver for 24" pin plus LC10 and Jx80 colour printers. (Please state 3.5" disc or M/D)@£23.00c
Upgrade to Sidewinder Plus@£8.05c

HEAT TRANSFER RIBBONS & PENS

Print your own T-shirt Design. Just print on paper and iron onto your T-shirt.
Star LC10NX1000, Rainbow@£17.25c Star LC10NX1000, Black@£11.50c Epson FX80, MX80, LX80, FX100/Okidata ML80 Citizen 120D/Star NX10, Black, Panasonic 1080/81@£11.50c Small 5 colour pen set@£12.65c Jumbo 5 Colour Pen Set@£16.10c

READYMADE LEADS

RGB QL to Phone@£5.75c RGB 8-6 pin DIN@£7.13c RGB 8-7 pin DIN (Hitachi)@£7.13c RGB 8-7 pin DIN (Ferguson)@£7.13c RGB 8 pin to SCART (Euro)@£11.50c 6-way PCC 25-way 'D' (Printer-Ser 1)@£9.89c

QPAC II

QPAC II Main menu windows adjust automatically to size. Files, directory, view, print, delete, backup, jobs, pick, Rjob, sort, channels, things exec, wake, buttons, Hotkey, Hotjobs. Fully multitasking, allows many programs to run at once. Requires min of 256k expanded memory@£49.91b Upgrade QRAM to QPAC II return master@£29.90b

AS REVIEWED QL WORLD
AUGUST ISSUE

CARE

OPEN
9am-5pm Mon-Thu
9am-4pm Fri

800 ST ALBANS ROAD,
GARSTON, WATFORD,
HERTS, WD2 6NL.
Tel: 0923-672102
Fax: 0923-662304

Please add carriage

a=£11.50 b=£3.45
c=£1.38 d=2.30

File Transfer: QL TO PC

Robin Stevenson joins the dots.

The transfer of text between the QL and a PC is a recurring need for many users, and there are even commercial programs to assist in this. The suggestion detailed here is a DIY system, to transfer text without having to leave your QL program such as *Quill*. This simplifies the process, and removes the need for temporary *_lis* files. As far as the program is concerned, it is printing the document. However, instead of the lead being plugged in to the printer, it will be attached to the serial socket of a PC.

There are three elements required: 1. A suitable lead. 2. A suitable printer driver. 3. MS-DOS commands to get the PC to receive the incoming data.

joined together. The simplest way to do this is to solder short sections of wire to each of these pins, and then solder the other ends together in one big, ungainly lump. Do this on a short extension to your printer lead, so that you can still use the original for your printer. The extension can also reverse the gender of the lead. A PC will require a female end to the lead, whereas most printers expect a male one. **Figure One** shows how you would make up a single, dedicated lead. **Figure Two** shows our hypothetical printer lead, plus the extension needed to make it suitable.

The Psion printer driver incorporates features which make it very useful for transfers. In particular the ability to specify any characters for bold, underline etc.

separate cartridge from *Quill*, you need only insert it into MDVL when you wish to transfer. The rest of the time, your normal printer driver will be used.

Finally there are the PC commands. To set up the PC's serial port, you need to enter

```
MODE COM1:1200,n,8,2
```

which is an MS-DOS external command. It configures serial port 1 with appropriate baud rate, parity, data bits and stop bits. Then, for every file you wish to transfer, before sending the print instruction, you need to enter

```
COPY COM1 FILENAME.TXT
```

where filename.txt is your own choice of name, including any appropriate path, and extension. A useful variant on this, while testing is

```
PY COM1 CON
```

which will send it to the PC's screen instead of to disk.

Figure One
Dedicated QL-to-PC lead.

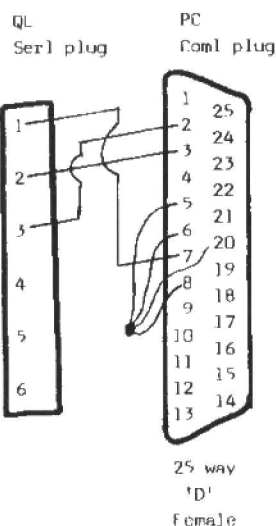


Figure Two - Extension to QL serial printer lead.

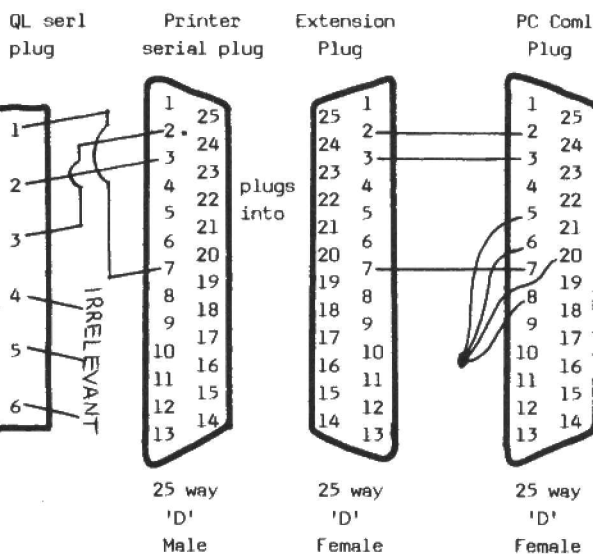


Figure Three
Psion printer driver, for
WordStar transfer format.

```
NAME      : QLtoPC
PORT      : ser1
BAUD RATE : 1200
PARITY    : none
CONT. FORMS : yes
END OF LINE : CR,LF
PREAMBLE  : none
POSTAMBLE : SUB
BOLD ON   : STX
BOLD OFF  : STX
UNDER ON  : DC3
UNDER OFF : DC3
SUBS. ON  : SYN
SUBS. OFF : SYN
SUPER. ON : DC4
SUPER. OFF : DC4
TRAN 1    : 'L,27,156,28
```

To start with the lead, getting the right connections is a manageable DIY job, provided you have access to a lead for the QL serial socket. Many users already have a ser1-to-25 pin serial printer lead. If so, the necessary adaptor can be made as a short extension to that. The only wires that need to go all the way through are the signal ground (pin 1 on the QL; pin 7 on the PC or printer plug), and the data lines, (pins 2 and 3 in both plugs). The data lines need to cross over, so that the QL's pin 2 goes to the PC's pin 3, and vice versa. In most printer leads, this cross-over will already be present, so the extension just carries the three wires directly across to their matching pins.

In addition, you need to convince the PC that something is about to happen. To do this, pins 5, 6, 8 and 20 all need to be

makes it possible to convert your embellishments to *WordStar* format, during the transfer. Any word-processor which allows importing of *WordStar* format text (such as *LocoScript PC*) will be able to make sense of these. It may be best to have no left hand, top or bottom margins, and no headers or footers. These can then be set by the receiving program.

Figure three shows a suggested printer driver, to produce *WordStar* compatible text. If you don't use the print enhancements any other word-processor should be able to use the result, if it can use or import standard Ascii files. The really vital settings are the baud rate, which must match the PC, and the Postamble, which must send the Control-Z value (Ascii 26, or SUB), to denote the end of the file. If you put this printer driver on a

Exactly the same technique should be usable from any other program. The same printer driver can be used to send *Archive* and *Abacus* reports, and provided you can set the baud rate, and send the Control-Z end-of-file marker, any other program that can print out could use the same technique.

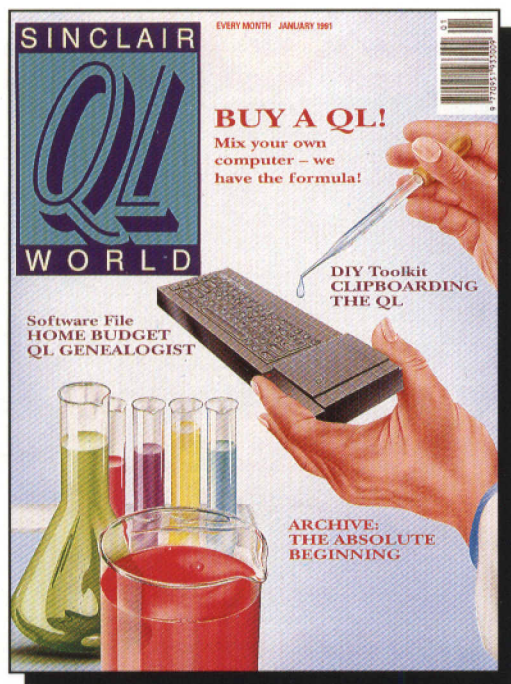
If you need to transfer QL files directly, rather than by printing them, you can do this by using the SuperBASIC command

```
BAUD 1200
```

After entering the MS-DOS COPY command, you can initiate the transfer from the QL with the line

```
COPY_N MDV2_FILENAME_EXT TO SERIZ
```


SUBSCRIBE TODAY



For
Convenience
To Avoid
Disappointment
And
Save Money

Direct to your door at
NO EXTRA COST*
— we pay the post!

Never again worry
about your local
newsagent selling
out.

Avoid any price
increases for the next
12 issues by
subscribing.

Simply fill in the coupon below and send it with your remittance to:
**M.S.M. Subs. Dept., Lazahold Ltd., P.O. Box 10, Roper Street,
Pallion Ind. Estate, Sunderland SR4 6SN.**

The first issue of a new subscription to be delivered will be one or two
issues after the one you placed your order in.

*Overseas prices include postage

Please send me one year's subscription to **Sinclair QL World**. I enclose my
cheque/money order for £..... made payable to MCPC LIMITED or
debit my Access/Visa card.

Card No.

Expiry date

Name

Address

Postcode (Please enter postcode to ensure prompt delivery)

Signed Date..... SQLW0291

**U.K.: £21.00, EUROPE: £32.00;
REST OF THE WORLD: £38.00**

AIRMAIL RATES AVAILABLE ON REQUEST